STC15F2K60S2系列单片机器件手册

---超强抗干扰，无法解密
---采用STC第八代加密技术
---不需要外部晶振和外部复位的单片机
---可省掉外部EEPROM，利用IAP技术
---ISP/IAP，在线编程，无需编程器/仿真器
---大容量2K字节SRAM
---双串口，两个独立串口
---高速10位A/D转换器，8通道
---1个时钟/机器周期8051
---高速，高可靠
---超低功耗，超低价
---超强抗静电，超强抗干扰

STC15F2K08S2	STC15L2K08S2
STC15F2K16S2	STC15L2K16S2
STC15F2K24S2	STC15L2K24S2
STC15F2K32S2	STC15L2K32S2
STC15F2K40S2	STC15L2K40S2
STC15F2K48S2	STC15L2K48S2
STC15F2K56S2	STC15L2K56S2
STC15F2K60S2	STC15L2K60S2
IAP15F2K61S2—本身就是仿真器—	IAP15L2K61S2

STC MCU Limited

采用STC第八代加密技术，现悬赏**10**万元人民币请专家帮忙查找加密有无漏洞

STC-ISP：最方便的在线升级软件, 无需专用编程器，无需专用仿真器

STC——8051单片机全球第一品牌，全球最大的8051单片机设计公司

请同行不要再抄袭我们的规格、设计和管脚排列，再抄袭就很无。。。

全部中国本土独立自主知识产权，请全体中国人民支持，您的支持是中国本土力量前进的有力保证.

封装后，全部175℃八小时高温烘烤，高品质制造保证

技术支持网站：www.STCMCU.com　www.GXWMCU.com

Update date: 2014/7/2

STC micro™
宏 晶 科 技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

STC15F2K60S2系列1T 8051单片机，2K字节SRAM，超高速双串口，高速A/D

不需外部晶振的单片机
不需外部复位的单片机
送仿真器
全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

采用宏晶第八代加密技术，现悬赏10万元人民币请专家帮忙查找加密有无漏洞

STC15F101W系列，大批量供货中，RMB0.99元起(STC15F100W)

SOP-8/DIP-8

封装形式：
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产但不推荐)
LQFP32(9mm x 9mm, 推荐)
SOP28(强烈推荐)
SKDIP28(正常生产)
TSSOP20(6.5mm x 6.5mm)

每片单片机具有全球唯一身份证号码(ID号)，
无法解密，加密坚不可摧
提供客制化IC设计服务

全部175℃
8小时高温烘烤

型号	工作电压(V)	Flash程序存储器(byte)	大容量SRAM字节	串行口	SPI仅有主机模式	普通定时器计数器T0-T2外部管脚也能掉电唤醒	CCP PCA PWM可当外部中断并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	所有封装 LQFP44/PDIP40 LQFP32 SOP28/SKDIP28/TSSOP20 (现此系列未生产PLCC44和SOP32封装) 部分封装价格(RMB ¥) TSSOP20	SOP28	SKDIP28	LQFP32	PDIP40	LQFP44
大批量现货供应中	STC15F2K60S2系列单片机选型价格一览表，另有STC15L系列(工作电压2.4V-3.6V) 特别提醒：3路CCP/PCA/PWM还可当3路定时器使用																								
STC15F2K08S2	5.5-4.2	8K	2K	2	有	3	3-ch	有	5	10位	2	53K	有	有	8级	有	是	有	是		¥3.8	¥4.0	¥3.9	¥4.5	¥4.0
STC15F2K16S2	5.5-4.2	16K	2K	2	有	3	3-ch	有	5	10位	2	45K	有	有	8级	有	是	有	是		¥4.0	¥4.2	¥4.1	¥4.7	¥4.2
STC15F2K32S2	5.5-4.2	32K	2K	2	有	3	3-ch	有	5	10位	2	29K	有	有	8级	有	是	有	是		¥4.3	¥4.5	¥4.4	¥4.9	¥4.5
STC15F2K40S2	5.5-4.2	40K	2K	2	有	3	3-ch	有	5	10位	2	21K	有	有	8级	有	是	有	是		¥4.4	¥4.6	¥4.5	¥4.9	¥4.6
STC15F2K48S2	5.5-4.2	48K	2K	2	有	3	3-ch	有	5	10位	2	13K	有	有	8级	有	是	有	是		¥4.4	¥4.6	¥4.5	¥4.9	¥4.6
STC15F2K56S2	5.5-4.2	56K	2K	2	有	3	3-ch	有	5	10位	2	5K	有	有	8级	有	是	有	是		¥4.4	¥4.6	¥4.5	¥4.9	¥4.6
STC15F2K60S2	5.5-4.2	60K	2K	2	有	3	3-ch	有	5	10位	2	1K	有	有	8级	有	是	有	是		¥4.4	¥4.6	¥4.5	¥4.9	¥4.6
IAP15F2K61S2 本身就是仿真器	5.5-4.2	61K	2K	2	有	3	3-ch	有	5	10位	2	IAP	有	有	8级	有	是	有	是	¥4.4	¥4.4	¥4.6	¥4.5	¥4.9	¥4.6
																				用户可将用户程序区的程序Flash当EEPROM使用					
IRC15F2K63S2 使用外部晶振或内部24MHz时钟	5.5-4.2	63.5K	2K	2	有	3	3-ch	有	5	10位	2	IAP	有	有	固定	有	是	无	否					¥4.9	¥4.6
																				用户可将用户程序区的程序Flash当EEPROM使用					
IAP15F2K61S	5.5-4.2	61K	2K	1	有	3	-	有	5	–	2	IAP	有	有	8级	有	是	有	是					¥4.5	¥4.0
																				用户可将用户程序区的程序Flash当EEPROM使用					
大批量供货中	STC15F101W系列单片机选型价格一览表，另有STC15L系列(工作电压2.4V-3.6V) 串行口功能可由[P3.0/INT4, P3.1]结合定时器实现																			SOP8	DIP8	DFN8			
STC15F100W	5.5-3.8	0.5K	128	-	-	2(T0/T2)	-	有	5	-	1	-	有	有	8级	有	是	有	是	¥0.99					
STC15F101W	5.5-3.8	1K	128	-	-	2(T0/T2)	-	有	5	-	1	4K	有	有	8级	有	是	有	是	¥1.2					
STC15F102W	5.5-3.8	2K	128	-	-	2(T0/T2)	-	有	5	-	1	3K	有	有	8级	有	是	有	是	¥1.3	¥1.5				
STC15F104W	5.5-3.8	4K	128	-	-	2(T0/T2)	-	有	5	-	1	1K	有	有	8级	有	是	有	是	¥1.4	¥1.6	¥1.5			
IRC15F107W 固定使用内部24MHz时钟	5.5-3.8	7K	128	-	-	2(T0/T2)	-	有	5	-	1	IAP	有	有	固定	有	是	无	否	¥1.4	¥1.6				
																				用户可将用户程序区的程序Flash当EEPROM使用					

以上单价为10K/M起定量，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

特别声明：以15F和15L开头且有SPI功能的芯片不支持"SPI从机模式"；以15W开头且有SPI功能的芯片，SPI主/从机模式均支持。

IRC15F2K63S2和IRC15L2K63S2型号单片机的内部复位门槛电压固定，P5.4不可当复位管脚RST使用，[XTAL2/P1.6, XTAL1/P1.7]不可当I/O口使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能。

STC 单片机在线编程/ISP及仿真线路(USB转串口)

建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]
大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

宏晶•STC15F2K60S2系列主要性能：

- 大容量2048字节片内RAM数据存储器
- 高速：1个时钟/机器周期，增强型8051内核，速度比传统8051快7～12倍
  速度也比STC早期的1T系列单片机(如STC12/11/10系列)的速度快20%
- 宽电压：5.5～4.2V，2.4～3.6V(STC15L2K60S2系列)
- 低功耗设计：低速模式，空闲模式，掉电模式(可由外部中断或内部掉电唤醒定时器唤醒)
- 不需外部复位的单片机，ISP编程时8级复位门槛电压可选，内置高可靠复位电路
- 不需外部晶振的单片机，ISP编程时内部时钟从5MHz~35MHz可设(相当于普通8051：60～420MHz)
  内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
- 支持掉电唤醒的资源有：INT0/INT1(上升沿/下降沿中断均可)，INT2/INT3/INT4(下降沿中断)；CCP0/CCP1/CCP2/T0/T1/T2管脚；内部掉电唤醒专用定时器
- 8/16/24/32/40/48/56/60/61/63.5K字节片内Flash程序存储器，擦写次数10万次以上
- 大容量片内EEPROM功能，擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- 高速ADC，8通道10位，速度可达30万次/秒，3路PWM还可当3路D/A使用
- 3通道捕获/比较单元(CCP/PCA/PWM)
  ----也可用来再实现3路D/A或3个定时器或3个外部中断(支持上升沿/下降沿中断)
- 利用CCP/PCA高速脉冲输出功能可实现3路9～16位PWM (每通道占用系统时间小于0.6%)
- 利用定时器T0、T1或T2的时钟输出功能可实现高精度的8～16位PWM (占用系统时间小于0.4%)
- 6个定时器，2个16位可重装载定时器T0和T1兼容普通8051的定时器，新增了一个16位的定时器T2，并都可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出(÷1或÷2或÷4)，3路CCP/PCA可再实现3个定时器
- 可编程时钟输出功能(对内部系统时钟或外部管脚的时钟输入进行时钟分频输出)：
  ① T0在P3.5输出时钟；② T1在P3.4输出时钟；③ T2在P3.0输出时钟，以上3个定时器/计数器输出时钟均可1~65536级分频输出；④ 内部主时钟在P5.4/MCLKO对外输出时钟(STC15系列8-pin单片机的主时钟在P3.4/MCLKO对外输出时钟)
- 超高速双串口/UART，两个完全独立的高速异步串行通信端口，分时切换可当5组串口使用
- SPI高速同步串行通信接口
- 硬件看门狗(WDT)
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(42/38/30/26/18个)，复位后为：准双向口/弱上拉(8051传统I/O口)
  可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
  每个I/O口驱动能力均可达到20mA，但整个芯片最大不要超过120mA
- 如果I/O口不够用可以用3根普通I/O口线外接74HC595(¥0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D作按键扫描来节省I/O口

选择宏晶•STC15F2K60S2系列单片机理由：

★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
★ 片内大容量2048字节SRAM
★ 无法解密，宏晶第八代加密技术，现悬赏10万元人民币请专家帮忙查找加密有无漏洞
★ 超强抗干扰：
　1. 高抗静电(ESD保护)整机轻松过2万伏静电测试　2. 轻松过4kV快速脉冲干扰(EFT测试)
　3. 宽电压，不怕电源抖动　4. 宽温度范围，-40℃~+85℃
★ 大幅降低EMI，内部可配置时钟，1个时钟/机器周期，
　可用低频时钟----出口欧美的有力保证
★ 超低功耗：
　1. 掉电模式：外部中断唤醒功耗 <0.1uA
　2. 空闲模式：典型功耗<1mA，　3. 正常工作模式：4mA～6mA
　4. 掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用于电池供电系统，如水表、气表等
★ 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元)，1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC15W1K16S 系列 1T 8051 单片机，宽电压，超高速串行口，比较器

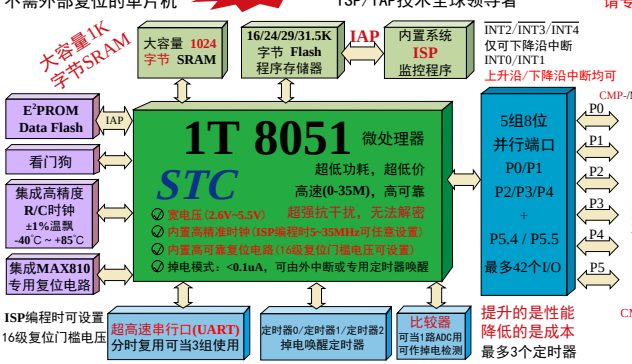
不需外部晶振的单片机
不需外部复位的单片机

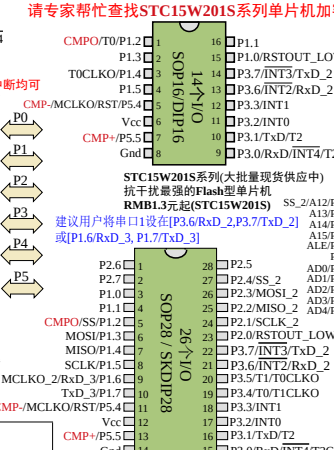
送仿真器

全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

采用宏晶第九代加密技术，现悬赏20万元人民币
请专家帮忙查找STC15W201S系列单片机加密有无漏洞

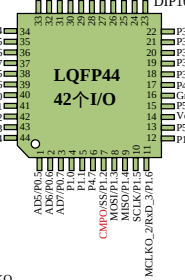
封装形式：
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产但不推荐)
LQFP32(9mm x 9mm, 推荐)
SOP28(强烈推荐)
SKDIP28(正常生产)
TSSOP20(6.5mm x 6.5mm)
SOP16(6mm x 9mm, 强烈推荐)
DIP16(正常生产)





STC15W201S系列(大批量现货供应中)
抗干扰最强的Flash型单片机
RMB1.3元起(STC15W201S)

建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]



全部175℃ 8小时高温烘烤

STC15W201S系列是目前全球抗干扰最强的Flash型单片机
现悬赏20万元人民币寻找比STC15W201S系列抗干扰更强的Flash型单片机

型号	工作电压(V)	Flash程序存储器(byte)	大容量SRAM字节	串行口	串行口可以STP1掉电唤醒	普通定时器计数器	CCP PCA PWM可当外部中断用	掉电唤醒专用定时器	标准外部中断4支持掉电唤醒	A/D(3路PWM可当3路D/A使用)	比较器(可当1路ADC使用,可作外部掉电检测)	EEPROM	内部低压检测中断	内部看门狗	内部高可靠复位(可选复位门槛电压)	可对内部时钟进行±1%调整	程序加密后防止反编译	支持RS485下载	封装
所有封装 LQFP44/PDIP40 LQFP32 SOP28/SKDIP28 TSSOP20 SOP16/DIP16																			
封装价格(RMB ¥)																			

STC15W201S系列单片机选型价格一览表，大批量现货供应中

																	SOP8	SOP16	DIP16	-	-
STC15W201S	2.4-5.5	1K	256	1	-	T0 / T2	–	有	5	–	有	4K	有	有	16级	有	是	有	是	¥1.3	¥1.4	¥1.6	-	-
STC15W202S	2.4-5.5	2K	256	1	-	T0 / T2	–	有	5	–	有	3K	有	有	16级	有	是	有	是	¥1.4	¥1.5	¥1.7	-	-
STC15W203S	2.4-5.5	3K	256	1	-	T0 / T2	–	有	5	–	有	2K	有	有	16级	有	是	有	是	¥1.5	¥1.6	¥1.8	-	-
STC15W204S	2.4-5.5	4K	256	1	-	T0 / T2	–	有	5	–	有	1K	有	有	16级	有	是	有	是	¥1.6	¥1.7	¥1.9	-	-
IAP15W205S	2.4-5.5	5K	256	1	-	T0 / T2	–	有	5	–	有	IAP	有	有	16级	有	是	有	是	¥1.6	¥1.7	¥1.9	-	-
IRC15W207S 固定使用内部24MHz时钟	2.4-5.5	7.5K	256	1	-	T0 / T2	–	有	5	–	有	IAP	有	有	固定	有	是	无	否	用户可将用户程序区的程序Flash当EEPROM使用				
																				¥1.6	¥1.7	¥1.9	-	-
																				用户可将用户程序区的程序Flash当EEPROM使用				

STC15W408S系列单片机选型价格一览表，大批量现货供应中

																	SOP28	SKDIP28	LQFP32	PDIP40	LQFP44
STC15W404S	2.4-5.5	4K	512	1	有	3	–	有	5	–	有	9K	有	有	16级	有	是	¥2.3	¥2.5	¥2.4	¥3.0	¥2.5
STC15W408S	2.4-5.5	8K	512	1	有	3	–	有	5	–	有	5K	有	有	16级	有	是	¥2.5	¥2.7	¥2.6	¥3.0	¥2.7
STC15W410S	2.4-5.5	10K	512	1	有	3	–	有	5	–	有	3K	有	有	16级	有	是					¥2.8
IAP15W413S	2.4-5.5	13K	512	1	有	3	–	有	5	–	有	IAP	有	有	16级	有	是	¥2.6	¥2.8	¥2.7	¥3.3	¥2.8
IRC15W415S 固定使用内部24MHz时钟	2.4-5.5	15.5K	512	1	有	3	–	有	5	–	有	IAP	有	有	固定	有	是	无	否	用户可将用户程序区的程序Flash当EEPROM使用				
																		¥2.6	¥2.8	¥2.7	¥3.3	¥2.8
																		用户可将用户程序区的程序Flash当EEPROM使用				

STC15W1K16S系列单片机选型价格一览表，大批量现货供应中

																	TSSOP20	SOP28	SKDIP28	LQFP32	PDIP40	LQFP44
STC15W1K16S	2.6-5.5	16K	1K	1	有	3	–	有	5	–	有	13K	有	有	16级	有	是	有	是	¥3.3	¥3.5	¥3.4	¥4.0	¥3.5
STC15W1K24S	2.6-5.5	24K	1K	1	有	3	–	有	5	–	有	5K	有	有	16级	有	是	有	是	¥3.6	¥3.6	¥3.8	¥3.7	¥4.3	¥3.8
IAP15W1K29S	2.6-5.5	29K	1K	1	有	3	–	有	5	–	有	IAP	有	有	16级	有	是	有	是	¥3.6	¥3.6	¥3.8	¥3.7	¥4.3	¥3.8
IRC15W1K31S 固定使用内部24MHz时钟	2.6-5.5	31.5K	1K	1	有	3	–	有	5	–	有	IAP	有	有	固定	有	是	无	否	用户可将用户程序区的程序Flash当EEPROM使用				
																				¥3.6	¥3.8	¥3.7	¥4.3	¥3.8
																				用户可将用户程序区的程序Flash当EEPROM使用				

以上单价为10K/M起定量，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

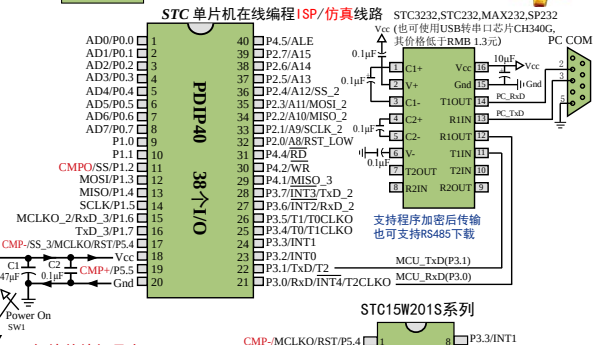
程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

上表中以IRC开头的单片机固定使用内部24MHz时钟，其内部复位门槛电压固定，P5.4不可当复位管脚RST使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能。

宏晶•STC15W1K16S系列主要性能：

大陆本土宏晶STC姚永平独立创新设计，请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

- 大容量1024字节片内RAM数据存储器
- 高速：1个时钟/机器周期，增强型8051内核，速度比传统8051快7～12倍
  速度也比STC早期的1T系列单片机(如STC12/11/10系列)的速度快20%
- 宽电压：2.6V ～ 5.5V
- 低功耗设计：低速模式，空闲模式，掉电模式(可由外部中断或内部掉电唤醒定时器唤醒)
- 不需外部复位的单片机，ISP编程时16级复位门槛电压可选，内置高可靠复位电路
- 不需外部晶振的单片机，ISP编程时内部时钟从5MHz~35MHz可设(相当于普通8051：60～420MHz)
  内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
- 支持掉电唤醒的资源有：INT0/INT1(上升沿/下降沿中断均可), INT2 / INT3 / INT4(下降沿中断)；
  RxD/T0/T1/T2管脚：内部掉电唤醒专用定时器
- 16/24/29K/31.5K字节片内Flash程序存储器，擦写次数10万次以上
- 大容量片内EEPROM功能，擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- 比较器，可当1路ADC使用，并可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出（可设置极性），也支持外部管脚CMP+与内部参考电压进行比较
- 3个16位可重装载定时器/计数器(T0/T1/T2，其中T0/T1兼容普通8051的定时器/计数器)，并都可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4）



每片单片机具有全球唯一身份证号码(ID号)，无法解密，加密坚不可摧

提供客制化IC设计服务

- 硬件看门狗(WDT)
- 超高速异步串行通信端口/UART, 分时切换可当3组串口使用SPI高速同步串行通信接口
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(42/38/30/26/18个)，**复位后为：准双向口/弱上拉**(8051传统I/O口)
  可设置四种模式：准双向口/弱上拉，**强推挽/强上拉，仅为输入/高阻，开漏**
  每个I/O口驱动能力均可达到20mA，但整个芯片最大不要超过120mA
- 如果I/O口不够用可以用3根普通I/O口线外接74HC595(¥0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D作按键扫描来节省I/O口

选择宏晶•STC15W1K16S系列单片机理由：

- ★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
- ★ 片内大容量1024字节SRAM
- ★ 无法解密，宏晶第九代加密技术，
  现悬赏20万元人民币请专家帮忙查找STC15W201S系列和STC15W1K16S系列加密有无漏洞
- ★ 超强抗干扰：
  1. 高抗静电(ESD保护) 整机轻松过2万伏静电测试
  2. 轻松过4kV快速脉冲干扰(EFT测试)
  3. 宽电压，不怕电源抖动
  4. 宽温度范围，-40℃~+85℃
- ★ 大幅降低EMI，内部可配置时钟，1个时钟/机器周期，
  可用低频时钟——出口欧美的有力保证
- ★ 超低功耗：
  1. **掉电模式：外部中断唤醒功耗** <0.1uA
  2. 空闲模式：典型功耗<1mA，
  3. 正常工作模式：4mA ~ 6mA
  4. **掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用于电池供电系统，如水表、气表、便携设备等**
- ★ **在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级**
- ★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元), 1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

宏晶•STC15W408AS 系列 1T 8051 单片机，超高速串口，高速A/D，比较器，宽电压

不需外部晶振的单片机
不需外部复位的单片机
送仿真器
全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

采用宏晶第九代加密技术，现悬赏**20**万元人民币请专家帮忙查找加密有无漏洞

封装形式：
SOP28(强烈推荐)
TSSOP28(6.4mm x 9.7mm)
SKDIP28(正常生产)
QFN28(5mm x 5mm)
SOP20(强烈推荐)
DIP20(正常生产)
TSSOP20 (6.5mm x 6.5mm)
SOP16(6mm x 9mm, 强烈推荐)
DIP16(正常生产)

全部175℃
8小时高温烘烤

建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]
或[P1.6/RxD_3, P1.7/TxD_3]

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器(可当1路ADC使用,可作掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	所有封装 SOP28/ SKDIP28 / TSSOP28 / QFN28 SOP20 / DIP20 / TSSOP20 SOP16 / DIP16 部分封装价格(RMB ¥) SOP16	DIP16	SOP20	DIP20	TSSOP20	SOP28	SKDIP28	TSSOP28	QFN28
STC15W408AS系列单片机选型价格一览表，大批量现货供应中 特别提示：3路CCP/PCA/PWM还可当3路定时器使用																													
STC15W401AS	2.4-5.5	1K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16级	有	是	有	是	¥1.60	¥1.80	¥1.80	¥2.00	¥1.90	¥2.00	¥2.20	¥2.10	¥2.20
STC15W402AS	2.4-5.5	2K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16级	有	是	有	是	¥1.80	¥2.00	¥2.00	¥2.20	¥2.10	¥2.20	¥2.40	¥2.30	¥2.40
STC15W404AS	2.4-5.5	4K	512	1	有	2	3-ch	有	5	10-bit	有	1	9K	有	有	16级	有	是	有	是	¥1.99	¥2.20	¥2.20	¥2.40	¥2.30	¥2.50	¥2.70	¥2.60	¥2.70
STC15W408AS	2.4-5.5	8K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16级	有	是	有	是	¥2.20	¥2.40	¥2.40	¥2.60	¥2.50	¥2.70	¥2.90	¥2.80	¥2.90
IAP15W413AS	2.4-5.5	13K	512	1	有	2	3-ch	有	5	10-bit	有	1	IAP	有	有	16级	有	是	有	是	¥2.30	¥2.50	¥2.50	¥2.70	¥2.60	¥2.80	¥3.00	¥2.90	¥3.00
																					用户可将用户程序区的程序Flash当EEPROM使用								
IRC15W415AS 使用外部晶振或内部24MHz时钟	2.4-5.5	15.5K	512	1	有	2	3-ch	有	5	10-bit	有	1	IAP	有	有	固定	有	是	无	否	¥2.30	¥2.50	¥2.50	¥2.70	¥2.60	¥2.80	¥3.00	¥2.90	¥3.00
																					用户可将用户程序区的程序Flash当EEPROM使用								

以上单价为10K/M起定量，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

IRC15W415AS型号单片机的内部复位门槛电压固定，P5.4不可当复位管脚RST使用，[XTAL2/P1.6, XTAL1/P1.7]不可当I/O口使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能 。

STC 单片机在线编程/**ISP及仿真**线路(*USB*转串口)

STC15F101W系列，大批量供货中
RMB0.99元起(STC15F100W)

SOP-8/DIP-8

大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无. . .

每片单片机具有
全球唯一身份证号码(ID号)
无法解密，加密坚不可摧

宏晶•STC15W408AS系列主要性能：

- 512字节片内RAM数据存储器
- 高速：1个时钟/机器周期，增强型8051内核，速度比传统8051快7～12倍
  速度也比STC早期的1T系列单片机(如STC12/11/10系列)的速度快20%
- 宽电压：2.4V ～ 5.5V
- 低功耗设计：低速模式，空闲模式，掉电模式(可由外部中断或内部掉电唤醒定时器唤醒)
- 不需外部复位的单片机，ISP编程时16级复位门槛电压可选，内置高可靠复位电路
- 不需外部晶振的单片机，ISP编程时内部时钟从5MHz~35MHz可设(相当于普通8051：60～420MHz)
  内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
- 支持掉电唤醒的资源有：INT0/INT1(上升沿/下降沿中断均可), INT2/INT3/INT4(下降沿中断)；
  CCP0/CCP1/CCP2/RxD/T0/T2管脚；内部掉电唤醒专用定时器
- 4K/8K/13K/15.5K字节片内Flash程序存储器，擦写次数10万次以上
- 大容量片内EEPROM功能, 擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程, 无需编程器/仿真器
- 高速ADC，8通道10位，速度可达30万次/秒，3路PWM还可当3路D/A使用
- 比较器，可当1路ADC使用，并可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出(可设置极性)，也支持外部管脚CMP+与内部参考电压进行比较
- 3通道捕获/比较单元(CCP/PCA/PWM)
  ----也可用来再实现3路D/A或3个定时器或3个外部中断(支持上升沿/下降沿中断)
- 利用CCP/PCA高速脉冲输出功能可实现3路9 ~ 16位PWM (每通道占用系统时间小于0.6%)
- 利用定时器T0的时钟输出功能可实现高精度的8 ~ 16位PWM (占用系统时间小于0.4%)
- 5个定时器，2个16位可重装载定时器/计数器(T0/T2，其中T0兼容普通8051的定时器/计数器)，并都可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出(÷1或÷2或÷4)，3路CCP/PCA可再实现3个定时器
- 可编程时钟输出功能(对内部系统时钟或外部管脚的时钟输入进行时钟分频输出)：
  ① T0在P3.5输出时钟；② T2在P3.0输出时钟，以上2个定时器/计数器输出时钟均可1~65536级分频输出；③ 内部主时钟在P5.4/MCLKO对外输出时钟(STC15系列8-pin单片机的主时钟在P3.4/MCLKO对外输出时钟)
- 硬件看门狗(WDT)
- 超高速异步串行通信端口/UART, 分时切换可当3组串口使用
- SPI高速同步串行通信接口
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(26/18/14个)，复位后为：准双向口/弱上拉(8051传统I/O口)
  可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
  每个I/O口驱动能力均可达到20mA，但整个芯片最大不要超过120mA

选择宏晶•STC15W408AS系列单片机理由：

★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
★ 无法解密，宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞
★ 超强抗干扰：
1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
2. 轻松过4kV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动
4. 宽温度范围，-40℃~+85℃
★ 大幅降低EMI，内部可配置时钟，1个时钟/机器周期，
可用低频时钟----出口欧美的有力保证
★ 超低功耗：
1. **掉电模式:外部中断唤醒功耗** <0.1uA
2. 空闲模式：典型功耗<1mA,　　3. 正常工作模式:4mA ~ 6mA
4. 掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用于电池供电系统，如水表、气表、便携设备等
★ 利用CCP/PCA高速脉冲输出功能可实现3路9 ~ 16位PWM (每通道占用系统时间小于0.6%)
★ 利用定时器T0的时钟输出功能可实现高精度的8 ~ 16位PWM (占用系统时间小于0.4%)
★ 在系统可仿真，在系统可编程，**无需专用编程器，无需专用仿真器，可远程升级**
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元), 1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，无法解密

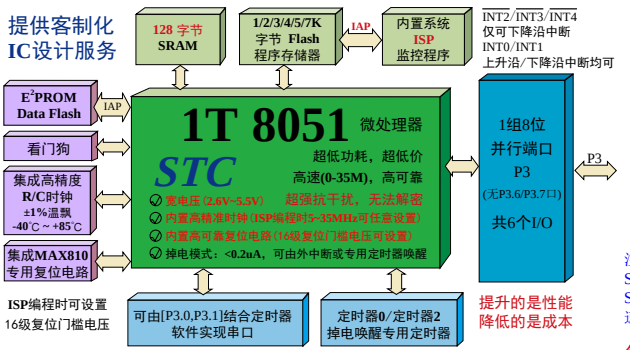
8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

宏晶•STC15W104系列新一代 1T 8051 单片机，宽电压，高可靠，超低价

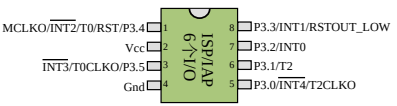
不需外部晶振的单片机
不需外部复位的单片机
送仿真器
全球第一款真正意义上的单片机ISP/IAP技术全球领导者
采用宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞

提供客制化IC设计服务



STC15W104系列，已开始大批量供货，RMB1.25元起(STC15W101)

全部175℃ 8小时高温烘烤



SOP-8/DIP-8/DFN-8

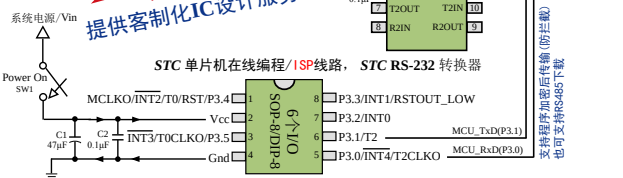
封装形式：
SOP-8/DIP-8/DFN-8

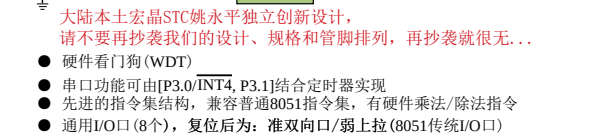
强烈推荐选择SOP-8贴片封装

注意：STC15W104系列与STC15F104E系列是不同的两个系列，STC15W104系列采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度比STC早期的1T系列单片机(如STC12/STC11/STC10系列)的速度快20%，而STC104E系列采用的是STC-Y3 CPU内核，速度没有STC15WF104系列的速度快。

每片单片机具有
全球唯一身份证号码(ID号)
无法解密，加密坚不可摧

大客户特价支持
提供客制化IC设计服务





大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口	SPI	普通定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM D/A	掉电唤醒定时器	标准外部中断支持掉电唤醒	A/D 8路	看门狗	内部低压检测中断	内部复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	封装8-Pin SOP-8	DIP-8	DFN-8
STC15W104系列单片机选型价格一览表，此系列大批量现货供应中 串行口功能可由[P3.0/INT4, P3.1]结合定时器实现																					
STC15W100	5.5-2.4	512	128	无	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.05	–	
STC15W101	5.5-2.4	1K	128	4K	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.25	–	
STC15W102	5.5-2.4	2K	128	3K	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.35	¥1.55	
STC15W104	5.5-2.4	4K	128	1K	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.45	¥1.65	¥1.55
IAP15W105	5.5-2.4	5K	128	IAP	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.45	¥1.65	¥1.55
IRC15W107 固定使用内部24MHz时钟	5.5-2.4	7K	128	IAP	无	无	2	无	有	5个	无	有	有	固定	有	是	无	否	¥1.45 用户可将用户程序区的程序FLASH当EEPROM使用	¥1.65 用户可将用户程序区的程序FLASH当EEPROM使用	
STC15F101W系列单片机选型价格一览表，该系列低压型号(工作电压2.4V-3.6V)建议用STC15W104系列代替 串行口功能可由[P3.0/INT4, P3.1]结合定时器实现 此系列大批量现货供应中																					
STC15F100W	5.5-3.8	512	128	无	无	无	2	无	有	5个	无	有	有	8级	有	是	有	是	¥0.99	–	
STC15F101W	5.5-3.8	1K	128	4K	无	无	2	无	有	5个	无	有	有	8级	有	是	有	是	¥1.2	–	
STC15F104W	5.5-3.8	4K	128	1K	无	无	2	无	有	5个	无	有	有	8级	有	是	有	是	¥1.4	¥1.6	¥1.5
IRC15F107W 固定使用内部24MHz时钟	5.5-3.8	7K	128	IAP	无	无	2	无	有	5个	无	有	有	固定	有	是	无	否	¥1.4 用户可将用户程序区的程序FLASH当EEPROM使用	¥1.6	

以上单价为10K/M起定量，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

IRC15W107型号单片机固定使用内部24Mhz时钟，其内部复位门槛电压固定，P5.4不可当复位管脚RST使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能。

宏晶•STC15W104系列主要性能：

- 高速：1个时钟/机器周期，增强型8051内核，速度比普通8051快6～12倍
  速度也比STC早期的1T系列单片机(如STC12/11/10系列)的速度快20%
- 宽电压：5.5～2.4V
- 低功耗设计：低速模式，空闲模式，掉电模式(可由外部中断或内部掉电唤醒定时器唤醒)
- 不需外部复位的单片机，ISP编程时16级复位门槛电压可选，内置高可靠复位电路
- 不需外部晶振的单片机，ISP编程时内部时钟从5MHz~35MHz可设(相当于8051：60～420MHz)
  内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
- 支持掉电唤醒的资源有：INT0/INT1(上升沿/下降沿中断均可), INT2/INT3/INT4(下降沿中断)；T0/T2管脚：内部掉电唤醒专用定时器
- 1K/2K/3K/4K/5K/7K字节片内Flash程序存储器，擦写次数10万次以上
- 128字节片内RAM数据存储器
- 片内EEPROM功能，擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- 2个16位可重装载定时器T0/T2，并可实现时钟输出功能，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4）
- 可编程时钟输出功能(对内部系统时钟或外部管脚的时钟输入进行时钟分频输出)：
  ① T0在P3.5输出时钟；② T2在P3.0输出时钟；③ 内部主时钟在P3.4/MCLKO对外输出时钟(STC15系列8-pin以上单片机的主时钟在P5.4/MCLKO对外输出时钟)
- 硬件看门狗(WDT)
- 串口功能可由[P3.0/INT4, P3.1]结合定时器实现
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(8个)，复位后为：准双向口/弱上拉(8051传统I/O口)
  可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
  每个I/O口驱动能力均可达到20mA，但整个芯片最大不要超过90mA
- 如果I/O口不够用可以用3根普通I/O口线外接74HC595(¥0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D作按键扫描来节省I/O口

选择宏晶•STC15W104系列单片机理由：

★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
★ 不需外部晶振的单片机，内部集成高精度R/C时钟(±0.3%)，
  ±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
★ 不需外部复位的单片机，内部集成高可靠复位电路，
  ISP编程时16级复位门槛电压可选，当然也可以继续用外部复位电路
★ 无法解密，宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞
★ 超强抗干扰：
  1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
  2. 轻松过4kV快速脉冲干扰(EFT测试)
  3. 宽电压，不怕电源抖动　　4. 宽温度范围，-40℃~+85℃
★ 大幅降低EMI，内部可配置时钟，1个时钟/机器周期，可用低频时钟
  ----出口欧美的有力保证
★ 超低功耗：
  1. 掉电模式：外部中断唤醒功耗 <0.2uA
  2. 空闲模式：典型功耗 <1mA,　　3. 正常工作模式：4mA -6mA
  4. 掉电模式可由外部中断唤醒，适用于电池供电系统，如水表、气表、便携设备等
★ 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元)，1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com

南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC15W4K60S4系列1T 8051单片机，4K字节SRAM，超高速四串口，6路15位PWM

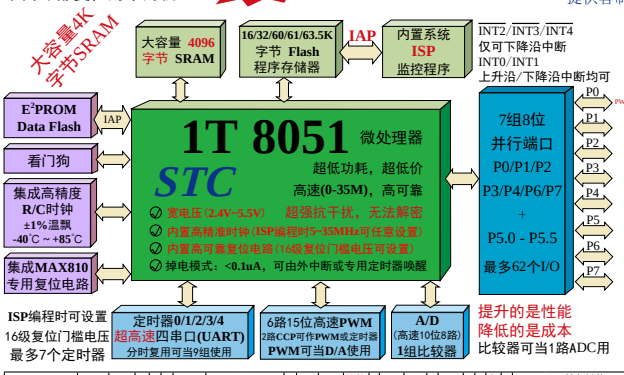
不需外部晶振的单片机
不需外部复位的单片机

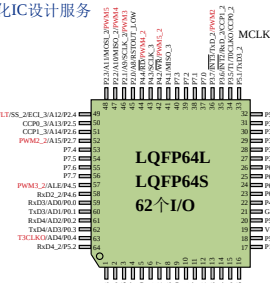
送仿真器

全球第一款真正意义上的单片机ISP/IAP技术全球领导者

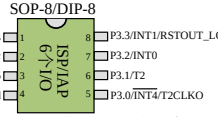
采用宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞

提供客制化IC设计服务





SOP-8/DIP-8

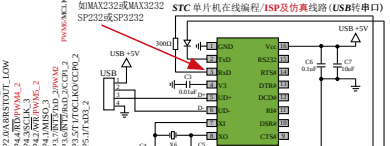


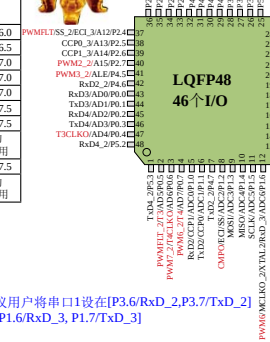
STC15F101W系列 大批量供货中
RMB1. 2元起 (STC15F101W)

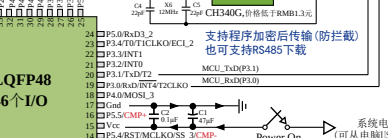
封装形式：
LQFP64L(16mm x 16mm)
LQFP64S(12mm x 12mm)
LQFP48(9mm x 9mm, 强烈推荐)
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产，但不推荐)
LQFP32(9mm x 9mm)
SOP28
SKDIP28
强烈推荐选择SOP28,
LQFP44/48/64贴片封装

全部175℃ 8小时高温烘烤

也可使用普通的RS232芯片接电脑
如MAX232或MAX3232 SP232或SP3232







大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无. . .

每片单片机具有全球
唯一身份证号码(ID号)，
无法解密，加密坚不可摧

型号	工作电压(V)	Flash程序存储器(byte)	大容量SRAM字节	串口可当SPI用	掉电唤醒	普通定时器	8路 PWM		掉电唤醒专用定时器	比较器(可当1路ADC使用，可作外部掉电检测)	A/D 8路(2路CCP可当3路D/A使用)	EEPROM	内部低压检测中断可掉电唤醒	看门狗	内部复位(可选8个门槛电压)	内部高精准时钟	时钟可外部输入或内部输出	程序加密后传输	支持RS485下载	所有封装 LQFP64/LQFP48/LQFP44/PDIP40 LQFP32/SOP28/SKDIP28 部分封装价格(RMB ¥)			
						T0-T4外部管脚也能掉电唤醒	6路15位专门的PWM(带死区控制)	2路CCP 10位PWM可当外部中断并可掉电唤醒												PDIP40	LQFP44	LQFP48	LQFP64
STC15W4K60S4系列单片机选型价格一览表，2014年8月开始供货 特别提醒：8路PWM可当8路D/A使用, 2路CCP可当2个定时器或2个外部中断使用																							
STC15W4K16S4	2.4-5.5	16K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	45K	有	有	16级	有	是	有	是	¥6.0	¥5.5	¥5.2	¥6.0
STC15W4K32S4	2.4-5.5	32K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	29K	有	有	16级	有	是	有	是	¥6.0	¥5.5	¥5.7	¥6.5
STC15W4K40S4	2.4-5.5	40K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	21K	有	有	16级	有	是	有	是	¥6.5	¥6.0	¥6.2	¥7.0
STC15W4K48S4	2.4-5.5	48K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	13K	有	有	16级	有	是	有	是	¥6.5	¥6.0	¥6.2	¥7.0
STC15W4K56S4	2.4-5.5	56K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	5K	有	有	16级	有	是	有	是	¥6.5	¥6.0	¥6.2	¥7.0
STC15W4K60S4	2.4-5.5	60K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	1K	有	有	16级	有	是	有	是	¥7.0	¥6.5	¥6.7	¥7.5
IAP15W4K61S4 本身就是仿真器	2.4-5.5	61K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	IAP	有	有	16级	有	是	有	是	¥7.0	¥6.5	¥6.7	¥7.5
																				用户可将用户程序区的程序Flash当EEPROM使用			
IRC15W4K63S4 使用外部晶振或内部24MHz时钟	2.4-5.5	63.5K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	IAP	有	有	16级	有	是	无	否	¥7.0	¥6.5	¥6.7	¥7.5
																				用户可将用户程序区的程序Flash当EEPROM使用			

以上单价为10K/M起定量，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

IRC15W4K63S4型号单片机的内部复位门槛电压固定，P5. 4不可当复位管脚RST使用，且P3. 2/P3. 3与下载无关，且不支持"程序加密后传输"功能。

建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]

宏晶•STC15W4K60S4系列主要性能：

- 大容量4096字节片内RAM数据存储器
- 高速：1个时钟/机器周期，增强型8051内核，速度比传统8051快7～12倍
  速度也比STC早期的1T系列单片机(如STC12/11/10系列)的速度快20%
- 宽电压：2.4V～5.5V
- 低功耗设计：低速模式，空闲模式，掉电模式(可由外部中断或内部掉电唤醒定时器唤醒)
- 不需外部复位的单片机，ISP编程时16级复位门槛电压可选，内置高可靠复位电路
- 不需外部晶振的单片机，ISP编程时内部时钟从5MHz~35MHz可设(相当于普通8051：60～420MHz)
  内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
- 支持掉电唤醒的资源有：INT0/INT1(上升沿/下降沿中断均可), INT2/INT3/INT4(下降沿中断)：CCP0/CCP1/RxD/RxD2/RxD3/RxD4/T0/T1/T2/T3/T4管脚；内部掉电唤醒专用定时器
- 16/32/40/48/56/60/61K/63.5K字节片内Flash程序存储器，擦写次数10万次以上
- 大容量片内EEPROM功能，擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- 高速ADC，8通道10位，速度可达30万次/秒，8路PWM还可当8路D/A使用
- 比较器，可当1路ADC使用，并可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出（可设置极性），也支持外部管脚CMP+与内部参考电压进行比较
- 6通道15位专门的高精度PWM(带死区控制)＋2通道CCP(利用它的高速脉冲输出功能可实现2路11~16位PWM)
  ---可用来再实现8路D/A，或2个16位定时器，或2个外部中断(支持上升沿/下降沿中断)
- 共7个定时器/计数器，5个16位可重装载定时器/计数器(T0/T1/T2/T3/T4,其中T0和T1兼容普通8051的定时器/计数器)，并均可实现时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4），2路CCP可再实现2个定时器
- 可编程时钟输出功能(对内部系统时钟或外部管脚的时钟输入进行时钟分频输出)：
  ① T0在P3.5输出时钟；② T1在P3.4输出时钟；③ T2在P3.0输出时钟；④ T3在P0.4输出时钟； ⑤ T4在P0.6输出时钟，以上5个定时器/计数器输出时钟均可1~65536级分频输出；⑥ 内部主时钟在P5.4/MCLKO对外输出时钟(STC15系列8-pin单片机的主时钟在P3.4/MCLKO对外输出时钟)
- 超高速四串口/UART，四个完全独立的高速异步串行通信端口，分时切换可当9组串口使用
- SPI高速同步串行通信接口
- 硬件看门狗(WDT)
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(62/46/42/38/30/26个)，复位后为：准双向口/弱上拉(8051传统I/O口)
  可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
  每个I/O口驱动能力均可达到20mA，但整个芯片最大不要超过120mA

选择宏晶•STC15W4K60S4系列单片机理由：

★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
★ 片内大容量4096字节SRAM
★ 6通道15位专门的高精度PWM(带死区控制)＋2通道CCP(利用它的高速脉冲输出功能可实现2路11~16位PWM)
---可用来再实现8路D/A,或2个16位定时器，或2个外部中断(支持上升沿/下降沿中断)
★ 无法解密，宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞
★ 超强抗干扰：
1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
2. 轻松过4kV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动
4. 宽温度范围，-40℃~+85℃
★ 大幅降低EMI，内部可配置时钟，1个时钟/机器周期，可用低频时钟
---出口欧美的有力保证
★ 超低功耗：
1. 掉电模式：外部中断唤醒功耗 <0.1uA
2. 空闲模式：典型功耗 <1mA，
3. 正常工作模式：4mA～6mA
4. 掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用于电池供电系统，如水表、气表等
★ 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元)，1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

目录

第1章 STC15全系列单片机总体介绍 23
1.1 STC15F2K60S2系列单片机总体介绍(C版大批量现货供应中) 23
1.1.1 STC15F2K60S2系列单片机简介 23
1.1.2 STC15F2K60S2系列单片机的内部结构图 26
1.1.3 STC15F2K60S2系列单片机管脚图 27
1.1.4 STC15F2K60S2系列单片机选型价格一览表 32
1.1.5 STC15F2K60S2系列单片机封装价格一览表 34
1.1.6 STC15F2K60S2系列单片机命名规则 35
1.1.7 STC15F2K60S2系列单片机最小应用系统 38
1.1.8 STC15F2K60S2系列接外部晶振和外部复位(实际不需要)的线路图 39
1.1.9 STC15F2K60S2系列在系统可编程(ISP)及仿真典型应用线路图 40
1.1.9.1 利用RS-232转换器的ISP下载编程典型应用线路图及仿真典型应用线路图 40
1.1.9.2 利用USB转串口的ISP下载编程典型应用线路图及仿真典型应用线路图 41
1.1.10 STC15F2K60S2系列管脚说明 42
1.2 STC15F101W系列单片机总体介绍(B版大批量现货供应中) 47
——STC15L101W系列建议用STC15W104系列取代 47
1.2.1 STC15F101W系列单片机简介(与STC15F104E系列有所不兼容) 47
1.2.2 STC15F101W系列单片机的内部结构图 49
1.2.3 STC15F101W系列单片机管脚图 50
1.2.4 STC15F101W系列单片机选型价格一览表 51
1.2.5 STC15F101W系列单片机命名规则 52
1.2.6 STC15F101W系列单片机在系统可编程(ISP)典型应用线路图 54
1.2.6.1 利用RS-232转换器的ISP下载编程典型应用线路图 54
1.2.6.2 利用USB转串口的ISP下载编程典型应用线路图 55
1.2.7 STC15F101W系列单片机的管脚说明 56
1.3 STC15W104系列单片机总体介绍(大批量现货供应中) 57
1.3.1 STC15W104系列单片机简介 57
1.3.2 STC15W104系列单片机的内部结构图 59
1.3.3 STC15W104系列单片机管脚图 60
1.3.4 STC15W104系列单片机选型价格一览表 61
1.2.5 STC15W104系列单片机命名规则 62
1.3.6 STC15W104系列单片机在系统可编程(ISP)典型应用线路图 64
1.3.6.1 利用RS-232转换器的ISP下载编程典型应用线路图 64
1.3.6.2 利用USB转串口的ISP下载编程典型应用线路图 65

1.3.7 STC15W104系列单片机的管脚说明 66
1.4 STC15W201S系列单片机总体介绍(A版大批量现货供应中) 67
1.4.1 STC15W201S系列单片机简介 67
1.4.2 STC15W201S系列单片机的内部结构图 69
1.4.3 STC15W201S系列单片机管脚图 70
1.4.4 STC15W201S系列单片机选型价格一览表 72
1.4.5 STC15W201S系列单片机命名规则 73
1.4.6 STC15W201S系列单片机在系统可编程(ISP)典型应用线路图 75
1.4.6.1 利用RS-232转换器的ISP下载编程典型应用线路图 75
1.4.6.2 利用USB转串口的ISP下载编程典型应用线路图 76
1.4.7 STC15W201S系列单片机的管脚说明 77
1.5 STC15W408AS系列单片机总体介绍(大批量现货供应中) 79
1.5.1 STC15W408AS系列单片机简介 79
1.5.2 STC15W408AS系列单片机的内部结构图 82
1.5.3 STC15W408AS系列单片机管脚图 83
1.5.4 STC15W408AS系列单片机选型价格一览表 87
1.4.5 STC15W408AS系列单片机封装价格一览表 87
1.5.6 STC15W408AS系列单片机命名规则 88
1.5.7 STC15W408AS系列单片机在系统可编程(ISP)典型应用线路图 90
1.5.7.1 利用RS-232转换器的ISP下载编程典型应用线路图 90
1.5.7.2 利用USB转串口的ISP下载编程典型应用线路图 91
1.5.8 STC15W408AS系列单片机的管脚说明 92
1.6 STC15W408S系列单片机总体介绍(大批量现货供应中) 95
1.6.1 STC15W408S系列单片机简介 95
1.6.2 STC15W408S系列单片机的内部结构图 98
1.6.3 STC15W408S系列单片机管脚图 99
1.6.4 STC15W408S系列单片机选型价格一览表 103
1.6.5 STC15W408S系列单片机封装价格一览表 104
1.6.6 STC15W408S系列单片机命名规则 104
1.6.7 STC15W408S系列单片机在系统可编程(ISP)典型应用线路图 106
1.6.7.1 利用RS-232转换器的ISP下载编程典型应用线路图 106
1.6.7.2 利用USB转串口的ISP下载编程典型应用线路图 107
1.6.8 STC15W408S系列单片机的管脚说明 108
1.7 STC15W1K16S系列单片机总体介绍(大批量现货供应中) 111
1.7.1 STC15W1K16S系列单片机简介 111
1.7.2 STC15W1K16S系列单片机的内部结构图 114
1.7.3 STC15W1K16S系列单片机管脚图 115
1.7.4 STC15W1K16S系列单片机选型价格一览表 119
1.7.5 STC15W1K16S系列单片机封装价格一览表 120

1.7.6 STC15W1K16S系列单片机命名规则 120
1.7.7 STC15W1K16S系列单片机在系统可编程(ISP)典型应用线路图 122
1.7.7.1 利用RS-232转换器的ISP下载编程典型应用线路图 122
1.7.7.2 利用USB转串口的ISP下载编程典型应用线路图 123
1.7.8 STC15W1K16S系列单片机的管脚说明 124
1.8 STC15W4K60S4系列单片机总体介绍(2014年8月开始送样) 127
1.8.1 STC15W4K60S4系列单片机简介 127
1.8.2 STC15W4K60S4系列单片机的内部结构图 130
1.8.3 STC15W4K60S4系列单片机管脚图 131
1.8.4 STC15W4K60S4系列单片机选型价格一览表 136
1.8.5 STC15W4K60S4系列单片机命名规则 137
1.8.6 STC15W4K60S4系列单片机在系统可编程(ISP)典型应用线路图 139
1.8.6.1 利用RS-232转换器的ISP下载编程典型应用线路图 139
1.8.6.2 利用USB转串口的ISP下载编程典型应用线路图 140
1.8.7 STC15W4K60S4系列单片机的管脚说明 141
1.9 STC15F408AD系列单片机总体介绍(大批量现货供应中) 148
1.9.1 STC15F408AD系列单片机简介 148
1.9.2 STC15F408AD系列单片机的内部结构图 151
1.9.3 STC15F408AD系列单片机管脚图 152
1.9.4 STC15F408AD系列单片机选型价格一览表 155
1.9.5 STC15F408AD系列单片机命名规则 156
1.9.6 STC15F408AD系列单片机在系统可编程(ISP)典型应用线路图 158
1.9.6.1 利用RS-232转换器的ISP下载编程典型应用线路图 158
1.9.6.2 利用USB转串口的ISP下载编程典型应用线路图 159
1.9.7 STC15F408AD系列单片机的管脚说明 160
1.10 STC15系列单片机封装尺寸图 163
1.10.1 DFN8封装尺寸图(不推荐，推荐选择SOP8封装) 163
1.10.2 SOP8封装尺寸图 164
1.10.3 DIP8封装尺寸图 165
1.10.4 SOP16封装尺寸图 166
1.10.5 DIP16封装尺寸图 167
1.10.6 SOP20封装尺寸图 168
1.10.7 TSSOP20封装尺寸图 169
1.10.8 LSSOP20封装尺寸图 170
1.10.9 DIP20封装尺寸图 171
1.10.10 SOP28封装尺寸图 172
1.10.11 TSSOP28封装尺寸图 173
1.10.12 SKDIP28封装尺寸图 174
1.10.13 QFN28封装尺寸图 175

1.10.14 LQFP32封装尺寸图 176
1.10.15 SOP32封装尺寸图 177
1.10.16 PDIP40封装尺寸图 178
1.10.17 LQFP44封装尺寸图 179
1.10.18 PLCC44封装尺寸图 180
1.10.19 PQFP44封装尺寸图 181
1.10.20 LQFP48封装尺寸图 182
1.10.21 LQFP64封装尺寸图 183
1.11 特殊外围设备(CCP/SPI, 串口1/2/3/4)在不同口间进行切换 185
1.11.1 CCP/PWM/PCA在多个口之间切换的测试程序(C和汇编) 187
1.11.2 PWM2/3/4/5/PWMFLT在多个口之间切换的测试程序(C和汇编) 189
1.11.3 PWM6/PWM7在多个口之间切换的测试程序(C和汇编) 191
1.11.4 SPI在多个口之间切换的测试程序(C和汇编) 193
1.11.5 串口1在多个口之间切换的测试程序(C和汇编) 195
1.11.6 串口2在多个口之间切换的测试程序(C和汇编) 197
1.11.7 串口3在多个口之间切换的测试程序(C和汇编) 199
1.11.8 串口4在多个口之间切换的测试程序(C和汇编) 201
1.12 每个单片机具有全球唯一身份证号码(ID号)及其测试程序 203
1.13 关于ID号在大批量生产中的应用方法(较多用户的用法) 209
1.14 在全球唯一ID号前添加软复位指令及重要测试参数 211
1.15 现供货的STC15系列中未实现的计划功能 213
1.15.1 现供货的STC15F2K60S2系列C版本中未实现的计划功能 213
1.15.1.1 现供货的STC15F2K60S2系列C版本的主时钟只可对外输出内部R/C时钟 213
——将在STC15W2K60S2系列中修正 213
1.15.1.2 现供货的STC15F2K60S2系列C版本的管脚RxD/RxD2不能唤醒掉电/停机模式 213
——将在STC15W2K60S2系列中修正 213
1.15.2 现供货的STC15F408AD系列C版本中未实现的计划功能 214
1.15.2.1 现供货的STC15F408AD系列C版本的RxD管脚不能唤醒掉电模式/停机模式 214
——将在STC15W408AS系列中修正 214
1.16 部分15系列单片机的特别注意事项 214
1.16.1 SPI的特别注意事项(仅针对以15F和15L开头的单片机) 214
——只支持SPI主机模式，不支持SPI从机模式 214
1.16.2 进入掉电唤醒模式的特别注意事项(仅针对以15L开头的单片机) 214
——以15L开头的单片机进入掉电模式前必须启动掉电唤醒定时器 214
1.16.3 STC15W201S系列A版本单片机的比较器下降沿中断不响应 215
——将在STC15W201S系列B版本中修正 215
1.16.4 STC15W408S及STC15W1K16S系列T0CLKO时钟输出功能的注意事 215
——若要使用T0CLKO时钟输出功能，须将P3.5口设置为强推挽输出 215

第2章 STC15系列的时钟、复位及省电模式 216
2.1 STC15系列单片机的时钟 216
2.1.1 STC15系列单片机的内部可配置时钟 217
2.1.2 主时钟分频和分频寄存器 218
2.1.3 可编程时钟输出(也可作分频器使用) 220
2.1.3.1 与可编程时钟输出有关的特殊功能寄存器 221
2.1.3.2 主时钟输出及测试程序(C和汇编) 226
2.1.3.3 定时器0对系统时钟或外部引脚T0的时钟输入进行可编程分频输出及测试程序 229
2.1.3.4 定时器1对系统时钟或外部引脚T1的时钟输入进行可编程分频输出及测试程序 233
2.1.3.5 定时器2对系统时钟或外部引脚T2的时钟输入进行可编程分频输出及测试程序 237
2.1.3.6 定时器3对系统时钟或外部引脚T3的时钟输入进行可编程分频输出及测试程序 241
2.1.3.7 定时器4对系统时钟或外部引脚T4的时钟输入进行可编程分频输出及测试程序 242
2.2 复位 243
2.2.1 外部RST引脚复位 243
2.2.2 软件复位及其测试程序(C和汇编) 243
2.2.3 掉电复位/上电复位 246
2.2.4 MAX810专用复位电路复位 246
2.2.5 内部低压检测复位 246
2.2.6 看门狗(WDT)复位 250
2.2.7 程序地址非法复位 254
2.2.8 热启动复位和冷启动复位 255
2.3 STC15系列单片机的省电模式 256
2.3.1 低速模式及其测试程序(C和汇编) 258
2.3.2 空闲模式(功耗<1mA)及其测试程序(C和汇编) 261
2.3.3 掉电模式/停机模式及其测试程序(C和汇编) 263
2.3.3.1 掉电模式/停机模式被唤醒后程序执行流程说明及测试程序(C和汇编) 268
2.3.3.2 用掉电唤醒专用定时器唤醒掉电模式/停机模式的测试程序(C和汇编) 271
——以15L开头的单片机进入掉电模式/停机模式前必须启动掉电唤醒专用定时器 271
2.3.3.3 用外部中断INT0(上升沿+下降沿)唤醒掉电模式/停机模式的测试程序(C和汇编) 273
2.3.3.4 用外部中断INT1(上升沿+下降沿)唤醒掉电模式/停机模式的测试程序(C和汇编) 275
2.3.3.5 用外部中断$\overline{\text{INT2}}$(下降沿)唤醒掉电模式/停机模式的测试程序(C和汇编) 277
2.3.3.6 用外部中断$\overline{\text{INT3}}$(下降沿)唤醒掉电模式/停机模式的测试程序(C和汇编) 279
2.3.3.7 用外部中断$\overline{\text{INT4}}$(下降沿)唤醒掉电模式/停机模式的测试程序(C和汇编) 281
2.3.3.8 用CCP/PCA扩展的外部中断(下降沿+上升沿)唤醒掉电模式/停机模式的程序 283
2.3.3.9 用RxD管脚由高到低的变化唤醒掉电模式/停机模式的测试程序(C和汇编) 288
——现供货的STC15F2K60S2系列C版本的RxD管脚不能唤醒掉电模式/停机模式 288
2.3.3.10 用RxD2管脚由高到低的变化唤醒掉电模式/停机模式的测试程序(C和汇编) 292
第3章 存储器和特殊功能寄存器(SFRs) 296
3.1 程序存储器 296
3.2 数据存储器(SRAM) 297

3.2.1 内部RAM 297
3.2.2 内部扩展RAM / XRAM / AUX-RAM及测试程序 299
3.2.3 使用内部扩展RAM的测试程序 302
3.2.3 外部64K数据总线 — 可外部扩展64K字节的数据存储器或外围设备 309
3.2.4 利用并行总线扩展外部32K SRAM的应用线路图 312
3.3 特殊功能寄存器(SFRs) 313
第4章 STC15系列单片机的I/O口结构 321
4.1 I/O口各种不同的工作模式及配置介绍 321
4.2 管脚P1.7/XTAL1与P1.6/XTAL2的特别说明 324
4.3 复位管脚RST的特别说明 324
4.4 管脚RSTOUT_LOW的特别说明 324
4.5 串行口1的中继广播方式 325
4.6 可将MCU从掉电模式/停机模式唤醒的外部管脚资源 326
4.7 与I/O口有关的特殊功能寄存器及其在程序中的地址声明 327
4.8 STC15系列单片机P0/P1/P2/P3/P4/P5口的测试程序 330
4.9 I/O口各种不同的工作模式结构框图 336
4.9.1 准双向口输出配置 336
4.9.2 强推挽输出配置 337
4.9.3 仅为输入（高阻）配置 337
4.9.4 开漏输出配置(若外加上拉电阻，也可读外部状态或输出高电平) 337
4.10 一种典型三极管控制电路 339
4.11 典型发光二极管控制电路 339
4.12 混合电压供电系统3V/5V器件I/O口互连 339
4.13 I/O口的外部输入何时低(0.8V以下)何时高电平(2.2V以上) 340
4.14 如何让I/O口上电复位时为低电平 341
4.15 PWM输出时I/O口的状态 341
4.16 I/O口行列式按键扫描应用线路图 342
4.17 74HC595管脚介绍及逻辑表 343
4.18 利用74HC595扩展I/O口的线路图(串行扩展，3根线) 344
4.19 利用74HC595驱动8个数码管(串行扩展, 3根线)的线路图 345
4.20 利用普通I/O口控制74HC595驱动8个数码管的测试程序 346
4.21 I/O口直接驱动LED数码管应用线路图 353
4.22 用STC MCU的I/O口直接驱动段码LCD的原理及扫描程序 354
4.23 A/D做按键扫描应用线路图 365

4.24 STC15系列单片机I/O口软件模拟I^2C接口的测试程序 366
4.24.1 STC15系列单片机I/O口软件模拟I2C接口的主机模式 366
4.24.2 STC15系列单片机I/O口软件模拟I2C接口的从机模式 370
第5章 指令系统 373
5.1 寻址方式 373
5.1.1 立即寻址 373
5.1.2 直接寻址 373
5.1.3 间接寻址 373
5.1.4 寄存器寻址 374
5.1.5 相对寻址 374
5.1.6 变址寻址 374
5.1.7 位寻址 374
5.2 完整指令集对照表(与传统8051对照) 375
——共111条指令，每条指令的详细执行时间 375
5.3 传统8051单片机指令定义详解(中文&English) 381
5.3.1 传统8051单片机指令定义详解 381
5.3.2 Instruction Definitions of Traditional 8051 MCU 421
第6章 中断系统 458
6.1 STC15系列单片机的中断请求源 459
6.1.1 STC15F101W系列单片机的中断请求源 459
6.1.2 STC15F408AD系列单片机的中断请求源 459
6.1.3 STC15W201S系列单片机的中断请求源 459
6.1.4 STC15W408AS系列单片机的中断请求源 460
6.1.5 STC15W408S系列单片机的中断请求源 460
6.1.6 STC15W1K16S系列单片机的中断请求源 460
6.1.7 STC15F2K60S2系列单片机的中断请求源 460
6.1.8 STC15W4K60S4系列单片机的中断请求源 460
6.2 中断结构图 461
6.3 中断向量入口地址/查询次序/优先级/请求标志/允许位表 464
6.4 在Keil C中如何声明中断函数 465
6.5 中断寄存器 466
6.6 中断优先级 475
6.7 中断处理 477
6.8 中断嵌套 478
6.9 外部中断 479

6.10 中断的测试程序(C和汇编)....480
6.10.1 外部中断0(INT0)的测试程序....480
6.10.1.1 外部中断INT0(上升沿+下降沿)的测试程序(C和汇编)....480
6.10.1.2 外部中断INT0(下降沿)的测试程序(C和汇编)....482
6.10.2 外部中断1(INT1)的测试程序....484
6.10.2.1 外部中断INT1(上升沿+下降沿)的测试程序(C和汇编)....484
6.10.2.2 外部中断INT1(下降沿)的测试程序(C和汇编)....486
6.10.3 外部中断2($\overline{\text{INT2}}$)(下降沿中断)的测试程序(C和汇编)....488
6.10.4 外部中断3($\overline{\text{INT3}}$)(下下降沿中断)的测试程序(C和汇编)....490
6.10.5 外部中断4($\overline{\text{INT4}}$)(下降沿中断)的测试程序(C和汇编)....492
6.10.6 T0扩展为外部下降沿中断的测试程序(C和汇编)....494
——利用T0的外部计数方式....494
6.10.7 T1扩展为外部下降沿中断的测试程序(C和汇编)....496
——利用T1的外部计数方式....496
6.10.8 T2扩展为外部下降沿中断的测试程序(C和汇编)....498
——利用T2的外部计数方式....498
6.10.9 用CCP/PCA功能扩展外部中断的测试程序(C和汇编)....501
第7章 定时器/计数器....505
7.1 定时器/计数器的相关寄存器....506
7.2 定时器/计数器0工作模式....515
7.2.1 模式0(16位自动重装载模式)及测试程序，建议只学习此模式足矣....515
——STC创新设计，请不要抄袭....515
7.2.1.1 定时器0的16位自动重装载模式的测试程序(C和汇编)....516
7.2.1.2 定时器0对系统时钟或外部引脚T0的时钟输入进行可编程分频输出的测试程序....519
——定时器0工作在16位自动重装载模式....519
7.2.1.3 T0的16位自动重装模式(软硬结合)模拟10位或16位PWM输出的程序(C和汇编)....522
7.2.1.4 T0的16位自动重装载模式扩展为外部下降沿中断的测试程序(C和汇编)....525
——利用T0的外部计数方式....525
7.2.2 模式1(16位不可重装载模式)，不建议学习....527
7.2.3 模式2(8位自动重装载模式)，不建议学习....528
7.2.4 模式3(不可屏蔽中断16位自动重装载,实时操作系统用节拍定时器)....531
7.3 定时器/计数器1工作模式....532
7.3.1 模式0(16位自动重装载模式)及测试程序，建议只学习此模式足矣....532
——STC创新设计，请不要抄袭....532
7.3.1.1 定时器1的16位自动重装载模式的测试程序(C和汇编)....533
7.3.1.2 定时器1对系统时钟或外部引脚T1的时钟输入进行可编程分频输出的测试程序....536
——定时器1工作在16位自动重装载模式....536
7.3.1.3 定时器1模式0(16位自动重载模式)作串口1波特率发生器的测试程序(C和汇编)....539
7.3.1.4 T1的16位自动重装载模式扩展为外部下降沿中断的测试程序(C和汇编)....544
——利用T1的外部计数方式....544

7.3.2 模式1(16位不可重装载模式)，不建议学习 546
7.3.3 模式2(8位自动重装载模式)，不建议学习 547
7.3.3.1 定时器1模式2(8位自动重载模式)作串口1波特率发生器的测试程序(C和汇编).....548
7.3.3.2 T1的8位自动重装载模式扩展为外部下降沿中断的测试程序(C和汇编).....553
7.4 古老的Intel 8051单片机定时器0/1应用举例.....555
7.5 定时器/计数器2及其应用(STC创新设计，请不要抄袭).....560
7.5.1 定时器/计数器2的相关特殊功能寄存器 560
7.5.2 定时器/计数器2作定时器及测试程序(C和汇编)..... 564
7.5.2.1 定时器2的16位自动重载模式的测试程序(C和汇编) 565
7.5.2.2 定时器2扩展为外部下降沿中断的的测试程序(C和汇编) 568
7.5.3 定时器2对系统时钟或外部引脚T2的时钟输入进行可编程分频输出 571
7.5.4 定时器/计数器2作串行口波特率发生器及测试程序(C和汇编)..... 575
7.5.4.1 定时器/计数器2作串行口1波特率发生器的测试程序(C和汇编) 576
7.5.4.2 定时器/计数器2作串行口2波特率发生器的测试程序(C和汇编) 582
7.6 定时器/计数器3及定时器/计数器4.....588
7.6.1 定时器/计数器3和定时器/计数器4的相关特殊功能寄存器 588
7.6.2 定时器/计数器3的应用(STC创新设计，请不要抄袭) 590
7.6.2.1 定时器/计数器3作定时器 590
7.6.2.2 定时器/计数器3对系统时钟或外部引脚T3的时钟输入进行可编程时钟分频输出 .. 591
7.6.2.3 定时器/计数器3作串行口3的波特率发生器 592
7.6.3 定时器/计数器4的应用(STC创新设计，请不要抄袭) 593
7.6.3.1 定时器/计数器4作定时器 593
7.6.3.2 定时器/计数器4对系统时钟或外部引脚T4的时钟输入进行可编程时钟分频输出 .. 594
7.6.3.3 定时器/计数器4作串行口4的波特率发生器 595
7.7 如何将定时器T0/T1/T2/T3/T4的速度提高12倍.....596
7.8 可编程时钟输出(也可作分频器使用) 598
7.8.1 与可编程时钟输出有关的特殊功能寄存器 599
7.8.2 主时钟输出及其测试程序(C和汇编)..... 604
7.8.3 定时器0对系统时钟或外部引脚T0的时钟输入进行可编程分频输出 607
——及测试程序(C和汇编)..... 607
7.8.4 定时器1对系统时钟或外部引脚T1的时钟输入进行可编程分频输出 611
——及测试程序(C和汇编)..... 611
7.8.5 定时器2对系统时钟或外部引脚T2的时钟输入进行可编程分频输出 615
——及测试程序(C和汇编)..... 615
7.8.6 定时器3对系统时钟或外部引脚T3的时钟输入进行可编程分频输出 619
——及测试程序(C和汇编)..... 619
7.8.7 定时器4对系统时钟或外部引脚T4的时钟输入进行可编程分频输出 620
——及测试程序(C和汇编)..... 620
7.8 掉电唤醒专用定时器及测试程序(C和汇编).....621

——进入掉电模式后可将单片机唤醒 621
——以15L开头的单片机进入掉电模式前必须启动掉电唤醒定时器 621
7.9 外部管脚T0/T1/T2/T3/T4如何唤醒掉电模式/停机模式 627
第8章 串行口通信 628
8.1 串行口1的相关寄存器 630
8.2 串行口1工作模式 637
8.2.1 串行口1工作模式0：同步移位寄存器(建议初学者不学) 637
8.2.2 串行口1工作模式1：8位UART，波特率可变 639
8.2.3 串行口1工作模式2：9位UART，波特率固定(建议不学习) 642
8.2.4 串行口1工作模式3：9位UART，波特率可变 644
8.3 串行口1的波特率设置 647
——串口1和串口2的波特率相同时,串口1和串口2可共享T2作波特率发生器 647
8.4 串行口1的测试程序(C和汇编) 652
8.4.1 定时器2作串口1波特率发生器的测试程序(C和汇编) 652
8.4.2 定时器1模式0(16位自动重装载)作串口1波特率发生器程序(C和汇编) 658
8.4.3 定时器1模式2(8位自动重装载)作串口1波特率发生器程序(建议不学) 663
8.5 串行口2的相关寄存器 668
8.6 串行口2工作模式 672
——串口2固定使用定时器T2作波特率发生器 672
——串口1/3/4和串口2的波特率相同时,串口1/3/4和串口2可共享T2作波特率发生器 672
8.6.1 串行口2的工作模式0----8位UART，波特率可变 672
8.6.2 串行口2的工作模式1----9位UART，波特率可变 672
8.7 串行口2的测试程序(C和汇编) 674
——使用定时器2作串口2的波特率发生器 674
8.8 串行口3的相关寄存器 680
8.9 串行口3工作模式 686
——串口3和串口2的波特率相同时,串口3和串口2可共享T2作波特率发生器 686
8.9.1 串行口3的工作模式0----8位UART，波特率可变 686
8.9.2 串行口3的工作模式1----9位UART，波特率可变 687
8.10 串行口4的相关寄存器 688
8.11 串行口4工作模式 694
——串口1和串口2的波特率相同时,串口1和串口2可共享T2作波特率发生器 694
8.11.1 串行口4的工作模式0----8位UART，波特率可变 694
8.11.2 串行口4的工作模式1----9位UART，波特率可变 695
8.12 双机通信 696

8.13 多机通信 707
8.14 串口1作为增强型串口使用时的自动地址识别功能 713
8.14.1 与串口1自动地址识别功能相关的特殊功能寄存器 713
8.14.2 串口1自动地址识别功能的介绍 716
8.14.3 串口1自动地址识别功能的测试程序(C和汇编) 718
8.15 串行口1的中继广播方式 724
8.16 用T0软件模拟串行口的测试程序(C及汇编) 725
——如串行口不够用或无串行口可用[P3.0, P3.1]结合定时器0软件模拟串行口 . 725
8.17 用T2结合INT4模拟一个半双工串口的测试程序(C及汇编) 734
8.18 利用两路CCP/PCA模拟一个全双工串口的程序(C及汇编) 744
第9章 STC15系列单片机EEPROM的应用 755
9.1 IAP及EEPROM新增特殊功能寄存器介绍 755
9.2 STC15系列单片机EEPROM空间大小及地址 759
9.2.1 STC15W4K60S4系列单片机EEPROM空间大小及地址 759
9.2.2 STC15F2K60S2及STC15L2K60S2系列EEPROM空间大小及地址 760
9.2.3 STC15W1K16S系列单片机EEPROM空间大小及地址 761
9.2.4 STC15W408S系列单片机EEPROM空间大小及地址 761
9.2.5 STC15W408AS系列单片机EEPROM空间大小及地址 762
9.2.6 STC15W201S系列单片机EEPROM空间大小及地址 762
9.2.7 STC15F408AD及STC15L408AD系列EEPROM空间大小及地址 763
9.2.8 STC15F101W及STC15L101W系列EEPROM空间大小及地址 763
9.3 IAP及EEPROM汇编简介 767
9.4 EEPROM测试程序(C和汇编) 771
9.4.1 EEPROM测试程序(不用串口送出数据)(C和汇编) 771
9.4.2 EEPROM测试程序(使用串口送出数据)(C和汇编) 779
9.5 比较器作外部掉电检测的参考电路 788
第10章 STC15系列单片机的A/D转换器 789
10.1 A/D转换器的结构 789
10.2 与A/D转换相关的寄存器 791
10.3 A/D转换典型应用线路 795
10.4 A/D作按键扫描应用线路图 795
10.5 A/D转换模块的参考电压源 797
10.6 A/D转换的测试程序(C和汇编) 798
10.6.1 A/D转换的测试程序(ADC中断方式) 798
10.6.2 A/D转换的测试程序(ADC查询方式) 804

10.7 利用新增的ADC第9通道测量内部参考电压的测试程序 811
——所测量的内部参考电压BandGap电压用来计算工作电压Vcc. 811
10.8 利用新增的ADC第9通道测量外部电压或外部电池电压 819
——利用内部参考电压BandGap电压测量 819
10.9 利用BandGap电压精确测量外部输入电压值及测试程序 820
10.10 利用SPI接口扩展12位ADC(TLC2543)的应用线路图 824
第11章 STC15系列CCP/PAC/PWM/DAC应用 825
11.1 与CCP/PWM/PCA应用有关的特殊功能寄存器 826
11.2 CCP/PWM/PCA模块的结构 834
11.3 CCP/PCA模块的工作模式 836
11.3.1 捕获模式 836
11.3.2 16位软件定时器模式 837
11.3.3 高速脉冲输出模式 838
11.3.4 脉宽调节模式(PWM) 839
11.3.4.1 8位脉宽调节模式(PWM) 839
11.3.4.2 7位脉宽调节模式(PWM)(STC创新设计，请不要抄袭) 841
11.3.4.3 6位脉宽调节模式(PWM)(STC创新设计，请不要抄袭) 842
11.4 用CCP/PCA功能扩展外部中断的测试程序(C和汇编) 844
11.5 用CCP/PCA功能实现16位定时器的测试程序(C和汇编) 848
11.6 CCP/PCA输出高速脉冲的测试程序(C和汇编) 852
11.7 CCP/PCA输出PWM(6位+7位+8位)的测试程序(C和汇编) 856
11.8 用CCP/PCA高速脉冲输出功能实现3路9~16位PWM的程序 860
——每通道占用系统时间小于0.6% 860
11.9 用CCP/PCA的16位捕获模式测脉冲宽度的程序(C和汇编) 875
11.10 用T0软硬结合模拟16路软件PWM的程序(C及汇编) 881
11.11 用T0的时钟输出功能实现8~16位PWM的程序(C及汇编) 888
——占用系统时间小于0.4% 888
11.12 用T1的时钟输出功能实现8~16位PWM的程序(C及汇编) 897
——占用系统时间小于0.4% 897
11.13 用T2的时钟输出功能实现8~16位PWM的程序(C及汇编) 905
——占用系统时间小于0.4% 905
11.14 利用两路CCP/PCA模拟一个全双工串口的程序(C及汇编) 913
11.15 比利用CCP/PCA模块实现8~16位DAC的参考线路图 924
第12章 STC15W系列的比较器 925

12.1 比较器中断方式程序举例(C及汇编) 928
12.2 比较器查询方式程序举例(C及汇编) 933
12.3 比较器作外部掉电检测的参考电路 936
12.4 STC15W系列比较器作ADC的程序举例(C语言) 937
12.5 在比较器负端产生不同的电压由比较器正端进行比较 943
12.6 现供货的STC15W201S系列A版本的比较器下降沿中断不响应 944
——将在STC15W201S系列B版本中修正 944
第13章 使用STC15系列单片机的ADC做电容感应触摸按键 945
第14章 同步串行外围接口(SPI接口) 967
14.1 与SPI功能模块相关的特殊功能寄存器 968
14.2 SPI接口的结构 972
14.3 SPI接口的数据通信 973
14.3.1 SPI接口的数据通信方式 974
14.3.2 对SPI进行配置 976
14.3.3 作为主机/从机时的额外注意事项 977
14.3.4 通过$\overline{SS}$改变模式 978
14.3.5 写冲突 978
14.3.6 数据模式 979
14.4 适用单主单从系统的SPI功能测试程序(C和汇编) 981
14.4.1 中断方式 981
14.4.2 查询方式 987
14.5 适用互为主从系统的SPI功能测试程序(C和汇编) 993
14.5.1 中断方式 993
14.5.2 查询方式 999
14.6 利用SPI控制74HC595驱动8位数码管及测试程序(C和汇编) 1005
14.7 利用SPI接口扩展12位ADC(TLC2543)的应用线路图 1015
14.8 利用STC15系列单片机SPI的主模式读写外部串行Flash 1016
14.8.1 利用STC15系列SPI的主模式读写外部串行Flash的参考电路图 1016
14.8.2 利用STC15系列SPI的主模式读写外部串行Flash的测试程序 1016
14.8.2.1 通过中断方式利用SPI的主模式读写外部串行Flash的测试程序(C和汇编) 1016
14.8.2.2 通过查询方式利用SPI的主模式读写外部串行Flash的测试程序(C和汇编) 1037
14.9 SPI的特别注意事项(仅针对以15F和15L开头的单片机) 1056
——只支持SPI主机模式，不支持SPI从机模式 1056
第15章 编译器(汇编器)/ISP编程器(烧录)/仿真器说明 1057
15.1 编译器/汇编器的说明及头文件 1057

15.2 ISP编程器/烧录器的说明 1069
15.2.1 在系统可编程(ISP)原理使用说明 1069
15.2.2 STC15系列在系统可编程(ISP)典型应用线路图..... 1070
15.2.2.1 利用RS-232转换器的ISP下载典型应用线路图..... 1070
15.2.2.2 利用USB转串口的ISP下载典型应用线路图 1072
15.2.3 所有STC系列单片机封装实物图..... 1073
15.2.4 STC-ISP下载编程工具硬件——STC-ISP下载板 1076
15.2.4.1 STC15系列ISP下载板实物图 1076
15.2.4.2 如何将单片机安装到STC-ISP下载板上..... 1077
15.2.4.3 如何使用转换座将贴片封装的单片机安装到STC-ISP下载板上..... 1078
15.2.4.3 如何将STC-ISP下载板连接到电脑..... 1083
15.2.5 针对USB-RS232转换线不兼容问题的几点说明 1085
15.2.6 如何用STC-ISP下载板给在用户系统上的单片机烧录用户程序 1086
15.2.7 电脑端的STC-ISP控制软件(Ver6.65)的界面使用说明..... 1088
15.2.8 STC-ISP控制软件(Ver6.65)发布项目程序使用说明..... 1096
15.2.9 “程序加密后传输”功能说明 1100
——防止烧录时通过串口分析出程序代码 1100
15.2.10 ″发布项目程序″+″程序加密后传输″结合使用 1104
15.2.11 运行用户程序时收到用户命令后自动启动ISP下载(不停电)..... 1111
15.2.12 用户接口 1113
15.2.13 RS485控制使用说明 1114
15.3 若无仿真器，如何调试/开发用户程序..... 1116
15.4 STC仿真器说明指南(建议串口放在P3.6/P3.7或P1.6/P1.7上)..... 1117
15.5 如何让传统的8051单片机学习板可仿真..... 1125
15.6 USB型联机/脱机下载工具U7/U7-S1 1127
15.6.1 USB型联机/脱机下载工具U7/U7-S1的功能介绍 1130
15.6.1.1 U7的功能介绍(U7的价格为人民币100元) 1130
15.6.1.2 U7-S1的功能介绍(U7-S1的价格为人民币50元)..... 1132
15.6.2 如何安装USB型联机/脱机下载工具U7/U7-S1的驱动程序 1134
15.6.3 U7/U7-S1的在线联机下载使用说明 1139
15.6.3.1 目标芯片直接安装于U7上的在线联机下载使用说明..... 1139
15.6.3.2 通过下载线连接目标芯片的U7/U7-S1的在线联机下载使用说明..... 1141
15.6.4 U7/U7-S1的脱机下载使用说明 1143
15.6.4.1 目标芯片直接安装于U7上时通过USB供电进行脱机下载的使用说明..... 1143
15.6.4.2 目标芯片通过下载线连接于U7/U7-S1时用USB供电进行脱机下载的使用说明 .. 1146
15.6.4.3 目标芯片直接安装于U7上时通过外部9V电源供电进行脱机下载的使用说明 ... 1148
15.6.4.4 目标芯片通过下载线连接于U7/U7-S1时用外部9V电源供电进行脱机下载的说明1150
15.6.4.5 通过用户板给简易脱机下载器U7/U7-S1供电进行脱机下载的使用说明 1152
15.6.4.6 U7/U7-S1与用户系统各自独立供电进行脱机下载的使用说明..... 1154
15.6.5 制作/更新USB型联机/脱机下载工具U7/U7-S1 1156

15.6.5.1 制作U7/U7-S1下载母片(U7控制母片) 1156
15.6.5.2 手动升级U7/U7-S1 1158
15.6.5.3 自动升级U7/U7-S1 1161
15.6.6 USB型联机/脱机下载板U7-S1的参考电路 1163
第16章 利用主控芯片对从芯片(限STC15系列)进行ISP下载 1167
附录A：STC15W104SW系列单片机总体介绍 1178
A.1 STC15W104SW系列简介(用STC15W201S系列取代) 1178
—— 管脚P1.0不要用，INT0/INT1/$\overline{\text{INT2}}$/$\overline{\text{INT2}}$/$\overline{\text{INT2}}$不要用 1178
A.2 STC15W104SW系列单片机的内部结构图 1180
A.3 STC15W104SW系列单片机管脚图 1181
A.4 STC15W104SW系列单片机选型价格一览表 1183
A.5 STC15W104SW单片机在系统可编程典型应用(ISP)线路图 1184
A.6 STC15W104SW系列单片机的管脚说明 1185
A.7 STC15W104SW系列单片机命名规则 1186
附录B：STC15系列单片机电气特性 1187
附录C：内部常规256字节RAM间接寻址测试程序 1188
附录D：用串口扩展I/O接口 1190
附录E：RS485自动控制或I/O口控制下载线路图 1193
附录F：一个I/O口驱动发光二极管并扫描按键 1194
附录G：STC15系列单片机取代传统8051注意事项 1195
附录H：STC15系列对指令系统的提升 1198
附录I：如何利用Keil C软件减少代码长度 1204
附录J：使用STC的IAP系列单片机开发自己的ISP程序 1205
——基于IAP15F2K61S2单片机 1205
附录K：STC15F2K60S2系列学习板电路示意图 1220
附录L：逻辑代数的基础 1223
——无微机原理的用户请从本章开始学习 1223
L.1 数制与编码 1223
L.1.1 数制转换 1223
L.1.2 原码、反码及补码 1226
L.1.3 常用编码 1226
L.2 几种常用的逻辑运算及其图形符号 1228

附录M：STC对单片机相关学科群部分课程改革呼吁 1231
附录N：STC对单片机课程教育改革的贡献 1233
附录O：STC推荐的单片机教材 1234
O.1 两本基于可仿真的STC15F2K60S2系列单片机的本科教材 1234
O.2 一本基于可仿真的STC15系列单片机的高职高专教材 1236
附录P：STC公司简介及招聘 1237
P.1 STC公司简介 1237
P.2 STC公司招聘 1237
附录Q：每日更新内容的备忘录 1238
附录R：STC彩色宣传资料 1241
R.1 STC15F2K60S2系列彩色宣传资料 1241
R.2 STC15W1K16S系列彩色宣传资料 1241
R.3 STC15W408AS系列彩色宣传资料 1241
R.4 STC15W104系列彩色宣传资料 1241
R.6 STC12C5A60S2系列彩色宣传资料 1241
R.7 STC11/10系列带外部数据总线的彩色宣传资料 1241
R.8 STC11/10系列无外部数据总线的彩色宣传资料 1241
R.9 STC12C5201AD系列彩色宣传资料 1241
R.10 STC12C5620AD系列彩色宣传资料 1241
R.11 STC12C5410AD/STC12C2052AD系列彩色宣传资料 1241
R.12 STC89C51/STC90C51系列彩色宣传资料 1241
R.13 STC15W4K60S4系列彩色宣传资料 1241

第1章 STC15全系列单片机总体介绍

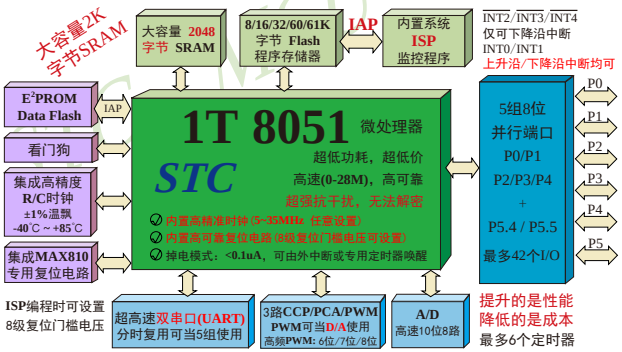
1.1 STC15F2K60S2系列单片机总体介绍(C版大批量现货供应中)

1.1.1 STC15F2K60S2系列单片机简介

STC15F2K60S2系列单片机是STC生产的单时钟/机器周期(1T)的单片机，是高速/高可靠/低功耗/超强抗干扰的新一代8051单片机，采用STC第八代加密技术，无法解密，指令代码完全兼容传统8051,但速度快8-12倍。内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP编程时8级复位门槛电压可选)。3路CCP/PWM/PCA，8路高速10位A/D转换(30万次/秒)，内置2K字节大容量SRAM，2组超高速异步串行通信端口(UART1/UART2，可在5组管脚之间进行切换，分时复用可作5组串口使用)，1组高速同步串行通信端口SPI，针对多串行口通信/电机控制/强干扰场合。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通8051快8-12倍
2. 工作电压：

   STC15F2K60S2 系列工作电压：5.5V - 4.2V（5V 单片机）
   STC15L2K60S2 系列工作电压：3.6V - 2.4V（3V 单片机）
3. 8K/16K/24K/32K/40K/48K/56K/60K/61K/63.5K字节片内Flash程序存储器，可擦写次数10万次以上
4. 片内大容量2048字节的SRAM，包括常规的256字节RAM <idata> 和内部扩展的1792字节 XRAM <xdata>
5. 大容量片内EEPROM，擦写次数10万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器，无需仿真器
7. 共8通道10位高速ADC，速度可达30万次/秒，3路PWM还可当3路D/A使用

8. 共3通道捕获/比较单元(CCP/PWM/PCA)
   ----也可用来再实现3个定时器或3个外部中断(支持上升沿/下降沿中断)或3路D/A
9. 利用CCP/PCA高速脉冲输出功能可实现3路9～16位PWM (每通道占用系统时间小于0.6%)
10. 利用定时器T0、T1或T2的时钟输出功能可实现高精度的8～16位PWM (占用系统时间小于0.4%)
11. 内部高可靠复位，ISP编程时8级复位门槛电压可选，可彻底省掉外部复位电路
12. 内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时内部时钟从5MHz~35MHz可设(5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz)
13. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
14. 工作频率范围：0MHz～28MHz，相当于普通8051的0MHz～336MHz
15. 两组超高速异步串行通信端口(可同时使用)，可在5组管脚之间进行切换，分时复用可当5组串口使用：
    串口1(RxD/P3.0, TxD/P3.1)可以切换到(RxD_2/P3.6, TxD_2/P3.7),
    还可以切换到(RxD_3/P1.6, TxD_3/P1.7)；
    串口2(RxD2/P1.0, TxD2/P1.1)可以切换到(RxD2_2/P4.6, TxD2_2/P4.7)
16. 一组高速异步串行通信端口SPI。
17. 支持程序加密后传输，防拦截
18. 支持RS485下载
19. 低功耗设计：低速模式，空闲模式，掉电模式/停机模式.
20. 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。
21. 可将掉电模式/停机模式唤醒的资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0 ($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚CCP0/CCP1/CCP2；管脚T0/T1/T2(下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；内部低功耗掉电唤醒专用定时器.
22. 共6个定时器/计数器，3个16位可重装载定时器/计数器(T0/T1/T2，其中T0/T1兼容普通8051的定时器/计数器)，并均可独立实现对外可编程时钟输出(3通道)，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4），3路CCP/PWM/PCA还可再实现3个定时器
23. 定时器/计数器T2，也可实现1个16位重装载定时器/计数器,T2也可产生时钟输出T2CLKO
24. 可编程时钟输出功能(对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出)：

STC MCU Limited

由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz.；
而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz.

① T0在P3.5/T0CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；

② T1在P3.4/T1CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出)；

③ T2在P3.0/T2CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；

以上3个定时器/计数器均可1~65536级分频输出。

④ 主时钟在P5.4/MCLKO对外输出时钟，并可如下分频MCLK/1, MCLK/2, MCLK/4.

现供货的STC15F2K60S2系列C版本单片机主时钟只可以对外输出内部R/C时钟，但是其他可外接外部晶体的STC15系列单片机的主时钟既可以对外输出内部R/C时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟。STC15F2K60S2系列下一升级版本——STC15W2K60S2系列单片机的主时钟将同其他系列单片机一样，其主时钟既可以对外输出内部R/C时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟。

上述MCLK是指主时钟频率，MCLKO是指主时钟输出。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列和STC15F2K60S2系列等)均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/P1.6口对外输出主时钟。

25. 硬件看门狗(WDT)

26. 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令

27. 通用I/O口（42/38/30/26个），复位后为：准双向口/弱上拉（普通8051传统I/O口）
可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
每个I/O口驱动能力均可达到20mA，但40-pin及40-pin以上单片机的整个芯片电流最大不要超过120mA，16-pin及以上/32-pin及以下单片机的整个芯片电流最大不要超过90mA.
如果I/O口不够用，可外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口

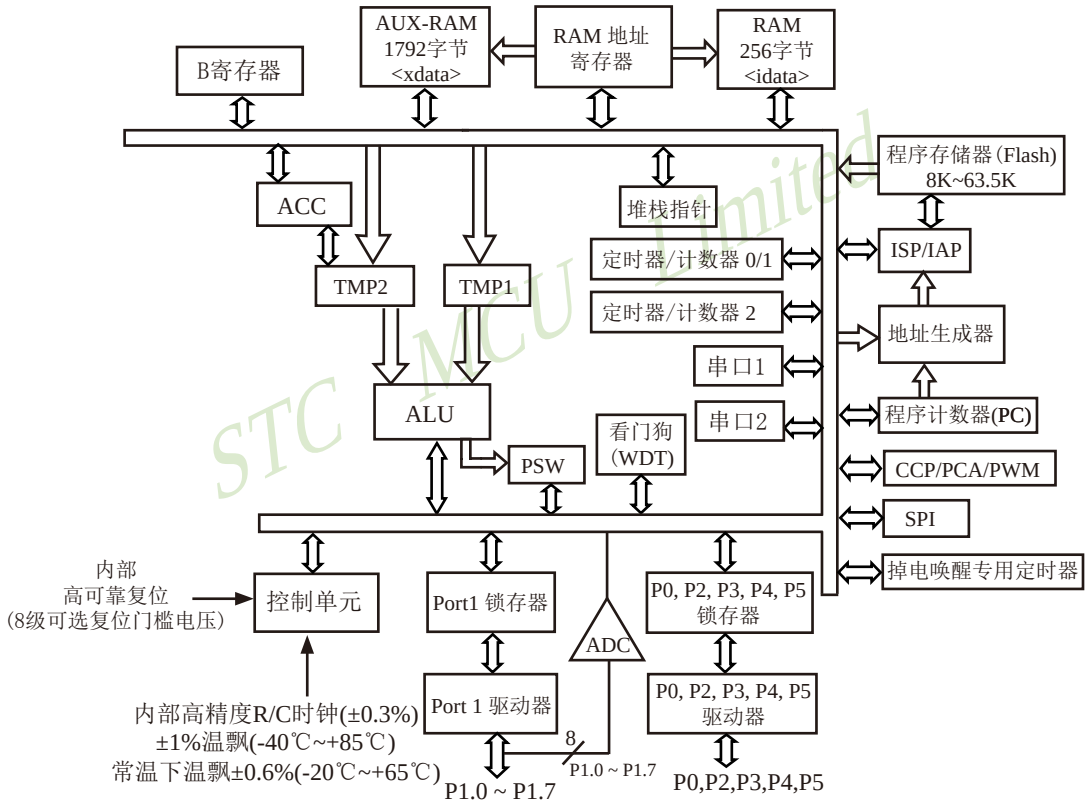
28. 封装：LQFP44(12mm x 12mm), LQFP32(9mm x 9mm), TSSOP20(6.5mm x 6.5mm), SOP28, SKDIP28, PDIP40.

29. 全部175℃八小时高温烘烤，高品质制造保证

30. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

1.1.2 STC15F2K60S2系列单片机的内部结构图

STC15F2K60S2系列单片机的内部结构框图如下图所示。STC15F2K60S2系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器、I/O口、高速A/D转换、看门狗、UART超高速异步串行通信口1/串行通信口2，CCP/PWM/PCA，1组高速同步串行端口SPI，片内高精度R/C时钟及高可靠复位等模块。STC15F2K60S2系列单片机几乎包含了数据采集和控制中所需的所有单元模块，可称得上是一个片上系统(SysTem Chip或SysTem on Chip,简写为STC,这是宏晶科技STC名称的由来)。



STC15F2K60S2系列内部结构框图

1.1.3 STC15F2K60S2系列单片机管脚图

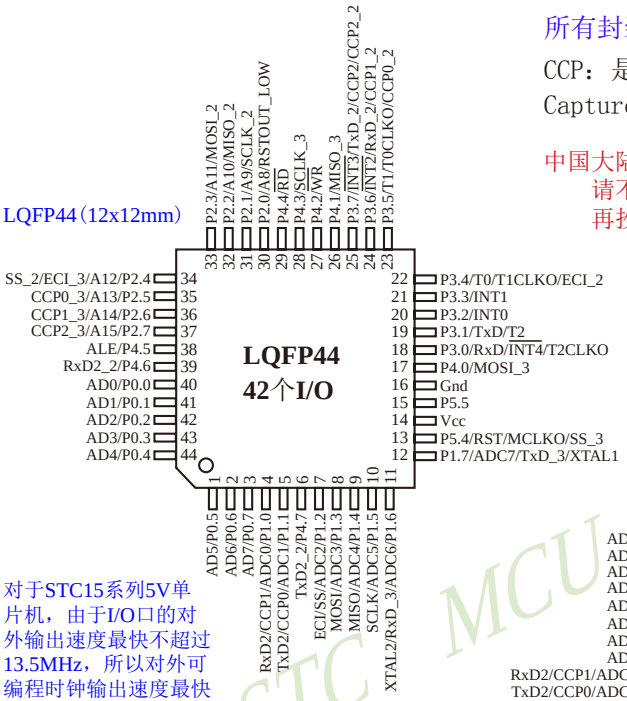
所有封装形式均满足欧盟RoHS要求，

CCP：是英文单词的缩写

Capture(捕获),Compare(比较),PWM(脉宽调制)

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

LQFP44(12x12mm)

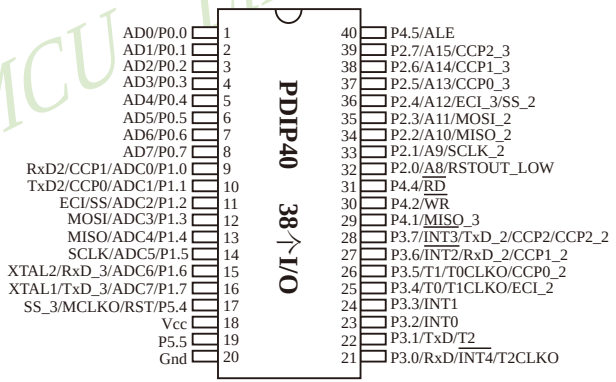


特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；



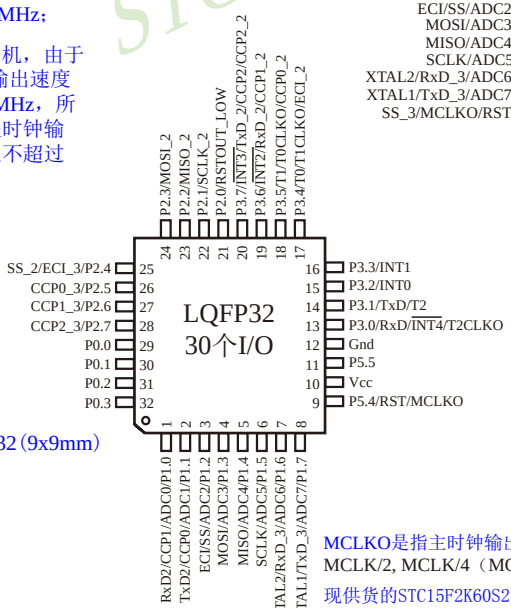
T0CLKO是指定时器/计数器0的可编程时钟输出(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；

T1CLKO是指定时器/计数器1的可编程时钟输出(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出)；

T2CLKO是指定时器/计数器2的可编程时钟输出(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；

T0CLKO/T1CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T1/T2的时钟输入进行时钟分频输出，作分频器使用。

LQFP32(9x9mm)



MCLKO是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4（MCLK是指主时钟频率）。

现供货的STC15F2K60S2系列C版本单片机的主时钟只可以是内部R/C时钟，STC15F2K60S2系列下一升级版本——STC15W2K60S2系列单片机主时钟既可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1]上。

STC MCU Limited

CCP：是英文单词的缩写
Capture(捕获),Compare(比较),PWM(脉宽调制)

A/D转换通道在P1口,管脚图中P1.x/ADCx是指P1.x管脚可作为A/D转换通道使用。

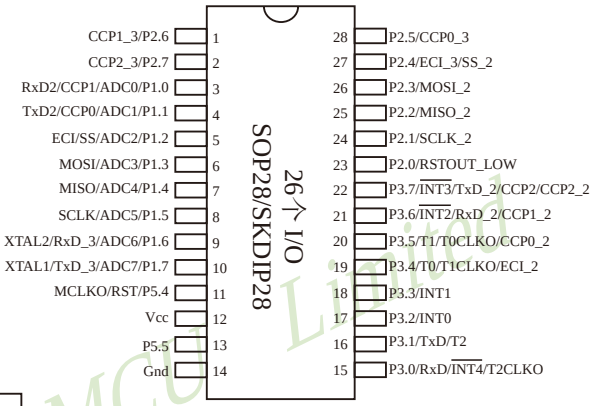
对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

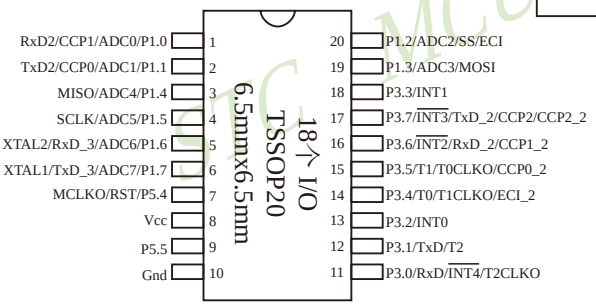
对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；

串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1]上。

MCLKO是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/**1, MCLK/2, MCLK/4**（MCLK是指主时钟频率）。

现供货的STC15F2K60S2系列C版本单片机的主时钟只可以是内部R/C时钟，STC15F2K60S2系列下一升级版本——STC15W2K60S2系列单片机主时钟既可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。





T0CLKO是指定时器/计数器0的可编程时钟输出(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出);
T1CLKO是指定时器/计数器1的可编程时钟输出(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出);
T2CLKO是指定时器/计数器2的可编程时钟输出(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出);

T0CLKO/T1CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T1/T2的时钟输入进行时钟分频输出，作分频器使用。

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,0000
P_SW2	BAH	Peripheral function switch						S4_S	S3_S	S2_S	xxxx,xxx0
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000,x000

串口1/S1可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD,P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2,P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1] 串口1在P1口时要使用内部时钟
1	1	无效

串口1建议放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1]上。

建议用户在程序中将[S1_S1, S1_S0]的值设置为[0, 1]或[1, 0]，进而将串口1放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1]上

CCP可在3个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP可在P1/P2/P3之间来回切换
0	0	CCP在[P1.2/ECI,P1.1/CCP0,P1.0/CCP1,P3.7/CCP2]
0	1	CCP在[P3.4/ECI_2,P3.5/CCP0_2,P3.6/CCP1_2,P3.7/CCP2_2]
1	0	CCP在[P2.4/ECI_3,P2.5/CCP0_3,P2.6/CCP1_3,P2.7/CCP2_3]
1	1	无效

SPI可在3个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI可在P1/P2/P4之间来回切换
0	0	SPI在[P1.2/SS,P1.3/MOSI,P1.4/MISO,P1.5/SCLK]
0	1	SPI在[P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI在[P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	无效

串口2/S2可在2个地方切换，由 S2_S 控制位来选择	
S2_S	S2可在P1/P4之间来回切换
0	串口2/S2在[P1.0/RxD2,P1.1/TxD2]
1	串口2/S2在[P4.6/RxD2_2,P4.7/TxD2_2]

DPS：DPTR registers select bit. DPTR 寄存器选择位

0: DPTR0 is selected　　DPTR0被选择

1: DPTR1 is selected　　DPTR1被选择

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000,x000

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， 输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， 输出时钟频率 = MCLK / 4

现供货的STC15F2K60S2系列C版本单片机的主时钟只可以是内部R/C时钟，STC15F2K60S2系列下一升级版本——STC15W2K60S2系列单片机主时钟既可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。上述MCLK是指主时钟频率。

STC15F2K60S2系列单片机在MCLKO/P5.4口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列、STC15F2K60S2系列等)均在MCLKO/P5.4口对外输出主时钟。

ADRJ：ADC转换结果调整

0：ADC_RES[7:0]存放高8位ADC结果，ADC_RESL[1:0]存放低2位ADC结果

1：ADC_RES[1:0]存放高2位ADC结果，ADC_RESL[7:0]存放低8位ADC结果

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

串口1的RxD管脚和TxD管脚可以在3组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1]；
[RxD_2/P3.6, TxD_2/P3.7]；
[RxD_3/P1.6, TxD_3/P1.7].

Tx2_Rx2：串口2的中继广播方式设置，功能暂时保留

串口2的RxD2管脚和TxD2管脚可以在2组不同管脚之间进行切换：[RxD2/P1.0, TxD2/P1.1]；
[RxD2_2/P4.6, TxD2_2/P4.7].

STC MCU Limited

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000,x000

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D转换的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

现供货的STC15F2K60S2系列C版本单片机的主时钟只可以是内部R/C时钟，STC15F2K60S2系列下一升级版本——STC15W2K60S2系列单片机主时钟既可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟

STC MCU Limited

1.1.4 STC15F2K60S2系列单片机选型价格一览表

型号	工作电压（V）	Flash程序存储器(byte)	大容量SRAM字节	串行口	SPI仅有主机模式	普通定时器计数器T0-T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	所有封装 LQFP44/PDIP40 LQFP32 SOP28/SKDIP28 TSSOP20 部分封装价格(RMB ¥) LQFP44	SOP28
STC15F2K60S2系列单片机选型价格一览表 特别提示：3路CCP/PCA/PWM还可当3路定时器使用																					
STC15F2K08S2	5.5-4.2	8K	2K	2	有	3	3-ch	有	5	10位	2	53K	有	有	8级	有	是	有	是	¥4.0	¥3.8
STC15F2K16S2	5.5-4.2	16K	2K	2	有	3	3-ch	有	5	10位	2	45K	有	有	8级	有	是	有	是	¥4.2	¥4.0
STC15F2K24S2	5.5-4.2	24K	2K	2	有	3	3-ch	有	5	10位	2	37K	有	有	8级	有	是	有	是	¥4.4	¥4.2
STC15F2K32S2	5.5-4.2	32K	2K	2	有	3	3-ch	有	5	10位	2	29K	有	有	8级	有	是	有	是	¥4.5	¥4.3
STC15F2K40S2	5.5-4.2	40K	2K	2	有	3	3-ch	有	5	10位	2	22K	有	有	8级	有	是	有	是	¥4.6	¥4.4
STC15F2K48S2	5.5-4.2	48K	2K	2	有	3	3-ch	有	5	10位	2	13K	有	有	8级	有	是	有	是	¥4.6	¥4.4
STC15F2K56S2	5.5-4.2	56K	2K	2	有	3	3-ch	有	5	10位	2	5K	有	有	8级	有	是	有	是	¥4.6	¥4.4
STC15F2K60S2	5.5-4.2	60K	2K	2	有	3	3-ch	有	5	10位	2	1K	有	有	8级	有	是	有	是	¥4.6	¥4.4
IAP15F2K61S2 本身就是仿真器	5.5-4.2	61K	2K	2	有	3	3-ch	有	5	10位	2	IAP	有	有	8级	有	是	有	是	¥4.6	¥4.4
																				用户可将用户程序区的程序FLASH当EEPROM使用	
IRC15F2K63S2 使用外部晶振或内部24MHz时钟	5.5-4.2	63.5K	2K	2	有	3	3-ch	有	5	10位	2	IAP	有	有	固定	有	是	无	否	¥4.6	–
																				用户可将用户程序区的程序FLASH当EEPROM使用	
IAP15F2K61S	5.5-4.2	61K	2K	1	有	3	-	有	5	–	2	IAP	有	有	8级	有	是	有	是	¥4.0	–
																				用户可将用户程序区的程序FLASH当EEPROM使用	
STC15F2K24AS	5.5-4.2	24K	2K	1	有	3	3-ch	有	5	10位	2	5K	有	有	8级	有	是	有	是	¥3.3	–
STC15L2K60S2系列单片机选型价格一览表																					
STC15L2K08S2	2.4-3.6	8K	2K	2	有	3	3-ch	有	5	10位	2	53K	有	有	8级	有	是	有	是	¥4.0	¥3.8
STC15L2K16S2	2.4-3.6	16K	2K	2	有	3	3-ch	有	5	10位	2	45K	有	有	8级	有	是	有	是	¥4.2	¥4.0
STC15L2K24S2	2.4-3.6	24K	2K	2	有	3	3-ch	有	5	10位	2	37K	有	有	8级	有	是	有	是	¥4.4	¥4.2
STC15L2K32S2	2.4-3.6	32K	2K	2	有	3	3-ch	有	5	10位	2	29K	有	有	8级	有	是	有	是	¥4.5	¥4.3
STC15L2K40S2	2.4-3.6	40K	2K	2	有	3	3-ch	有	5	10位	2	22K	有	有	8级	有	是	有	是	¥4.6	¥4.4
STC15L2K48S2	2.4-3.6	48K	2K	2	有	3	3-ch	有	5	10位	2	13K	有	有	8级	有	是	有	是	¥4.6	¥4.4
STC15L2K56S2	2.4-3.6	56K	2K	2	有	3	3-ch	有	5	10位	2	5K	有	有	8级	有	是	有	是	¥4.6	¥4.4
STC15L2K60S2	2.4-3.6	60K	2K	2	有	3	3-ch	有	5	10位	2	1K	有	有	8级	有	是	有	是	¥4.6	¥4.4
IAP15L2K61S2 本身就是仿真器	2.4-3.6	61K	2K	2	有	3	3-ch	有	5	10位	2	IAP	有	有	8级	有	是	有	是	¥4.6	¥4.4
																				用户可将用户程序区的程序FLASH当EEPROM使用	
IAP15L2K61S	2.4-3.6	61K	2K	1	有	3	-	有	5	–	2	IAP	有	有	8级	有	是	有	是	¥4.0	–
																				用户可将用户程序区的程序FLASH当EEPROM使用	

STC MCU Limited

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

注意：因为程序区的最后7个字节单元被强制性的放入全球唯一ID号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少7个字节。

IRC15F2K63S2和IRC15L2K63S2型号单片机的内部复位门槛电压固定，P5.4不可当复位管脚RST使用，[XTAL2/P1.6, XTAL1/P1.7]不可当I/O口使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能。

特别声明：以15F和15L开头且有SPI功能的芯片,只支持"SPI主机模式"，不支持"SPI从机模式"；以15W开头且有SPI功能的芯片，SPI主/从机模式均支持

特别声明：以15L开头的C版本芯片如需进入"掉电模式",进入"掉电模式"前必须启动掉电唤醒定时器<3uA>,不超过1秒要唤醒一次，以15F和15W开头的芯片以及最新的以15L开头的D版本芯片则不需要

总结：STC15F/L2K60S2系列单片机(含IRC15F2K63S2型号单片机)有：3个16位可重装载普通定时器/计数器，这3个普通定时器/计数器分别是定时器/计数器0、定时器/计数器1和定时器/计数器2；3路CCP/PWM/PCA(可再实现3个定时器或3个D/A转换器)；掉电唤醒专用定时器；5个外部中断INT0/INT1/$\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$；2组高速异步串行口(可同时使用)；1组高速同步串行通信端口SPI；8路高速10位A/D转换器；2个数据指针DPTR；外部数据总线等功能。

IAP15F/L2K61S型号单片机有：3个16位可重装载普通定时器/计数器，这3个普通定时器/计数器分别是定时器/计数器0、定时器/计数器1和定时器/计数器2；掉电唤醒专用定时器；5个外部中断INT0/INT1/$\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$；1组高速异步串行口；1组高速同步串行通信端口SPI；2个数据指针DPTR；外部数据总线等功能。

STC15F2K24AS型号单片机有：3个16位可重装载普通定时器/计数器，这3个普通定时器/计数器分别是定时器/计数器0、定时器/计数器1和定时器/计数器2；3路CCP/PWM/PCA(可再实现3个定时器或3个D/A转换器)；掉电唤醒专用定时器；5个外部中断INT0/INT1/$\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$；1组高速异步串行口；1组高速同步串行通信端口SPI；8路高速10位A/D转换器；2个数据指针DPTR；外部数据总线等功能。

STC MCU Limited

1.1.5 STC15F2K60S2系列单片机封装价格一览表

型号	工作电压（V）	工作频率(MHz)	工作温度(I — 工业级)	所有封装价格(RMB ¥) LQFP44 / PDIP40 LQFP32 SOP28 / SKDIP28 TSSOP20					
				LQFP44	PDIP40	LQFP32	SOP28	SKDIP28	TSSOP20
STC15F2K60S2系列单片机封装价格一览表									
STC15F2K08S2	5.5-4.2	28	-40℃ ~ +85℃	¥4.0	¥4.5	¥3.9	¥3.8	¥4.0	
STC15F2K16S2	5.5-4.2	28	-40℃ ~ +85℃	¥4.2	¥4.7	¥4.1	¥4.0	¥4.2	
STC15F2K24S2	5.5-4.2	28	-40℃ ~ +85℃	¥4.4	¥4.9	¥4.3	¥4.2	¥4.4	
STC15F2K32S2	5.5-4.2	28	-40℃ ~ +85℃	¥4.5	¥4.9	¥4.4	¥4.3	¥4.5	
STC15F2K40S2	5.5-4.2	28	-40℃ ~ +85℃	¥4.6	¥4.9	¥4.5	¥4.4	¥4.6	
STC15F2K48S2	5.5-4.2	28	-40℃ ~ +85℃	¥4.6	¥4.9	¥4.5	¥4.4	¥4.6	
STC15F2K56S2	5.5-4.2	28	-40℃ ~ +85℃	¥4.6	¥4.9	¥4.5	¥4.4	¥4.6	
STC15F2K60S2	5.5-4.2	28	-40℃ ~ +85℃	¥4.6	¥4.9	¥4.5	¥4.4	¥4.6	
IAP15F2K61S2 本身就是仿真器	5.5-4.2	28	-40℃ ~ +85℃	¥4.6	¥4.9	¥4.5	¥4.4	¥4.6	¥4.4
IRC15F2K63S2 使用外部晶振或内部24MHz时钟	5.5-4.2	28	-40℃ ~ +85℃	¥4.6	¥4.9	无	无	无	
IAP15F2K61S	5.5-4.2	28	-40℃ ~ +85℃	¥4.0	¥4.5	无	无	无	
STC15F2K24AS	5.5-4.2	28	-40℃ ~ +85℃	¥3.3	无	无	无	无	
STC15L2K60S2系列单片机封装价格一览表									
STC15L2K08S2	2.4-3.6	28	-40℃ ~ +85℃	¥4.0		¥3.9	¥3.8		
STC15L2K16S2	2.4-3.6	28	-40℃ ~ +85℃	¥4.2		¥4.1	¥4.0		
STC15L2K24S2	2.4-3.6	28	-40℃ ~ +85℃	¥4.4		¥4.3	¥4.2		
STC15L2K32S2	2.4-3.6	28	-40℃ ~ +85℃	¥4.5		¥4.4	¥4.3		
STC15L2K40S2	2.4-3.6	28	-40℃ ~ +85℃	¥4.6		¥4.5	¥4.4		
STC15L2K48S2	2.4-3.6	28	-40℃ ~ +85℃	¥4.6		¥4.5	¥4.4		
STC15L2K56S2	2.4-3.6	28	-40℃ ~ +85℃	¥4.6		¥4.5	¥4.4		
STC15L2K60S2	2.4-3.6	28	-40℃ ~ +85℃	¥4.6		¥4.5	¥4.4		
IAP15L2K61S2 本身就是仿真器	2.4-3.6	28	-40℃ ~ +85℃	¥4.6	¥4.9	¥4.5	¥4.4	¥4.6	
IAP15L2K61S	2.4-3.6	28	-40℃ ~ +85℃	¥4.0	无	无	无	无	

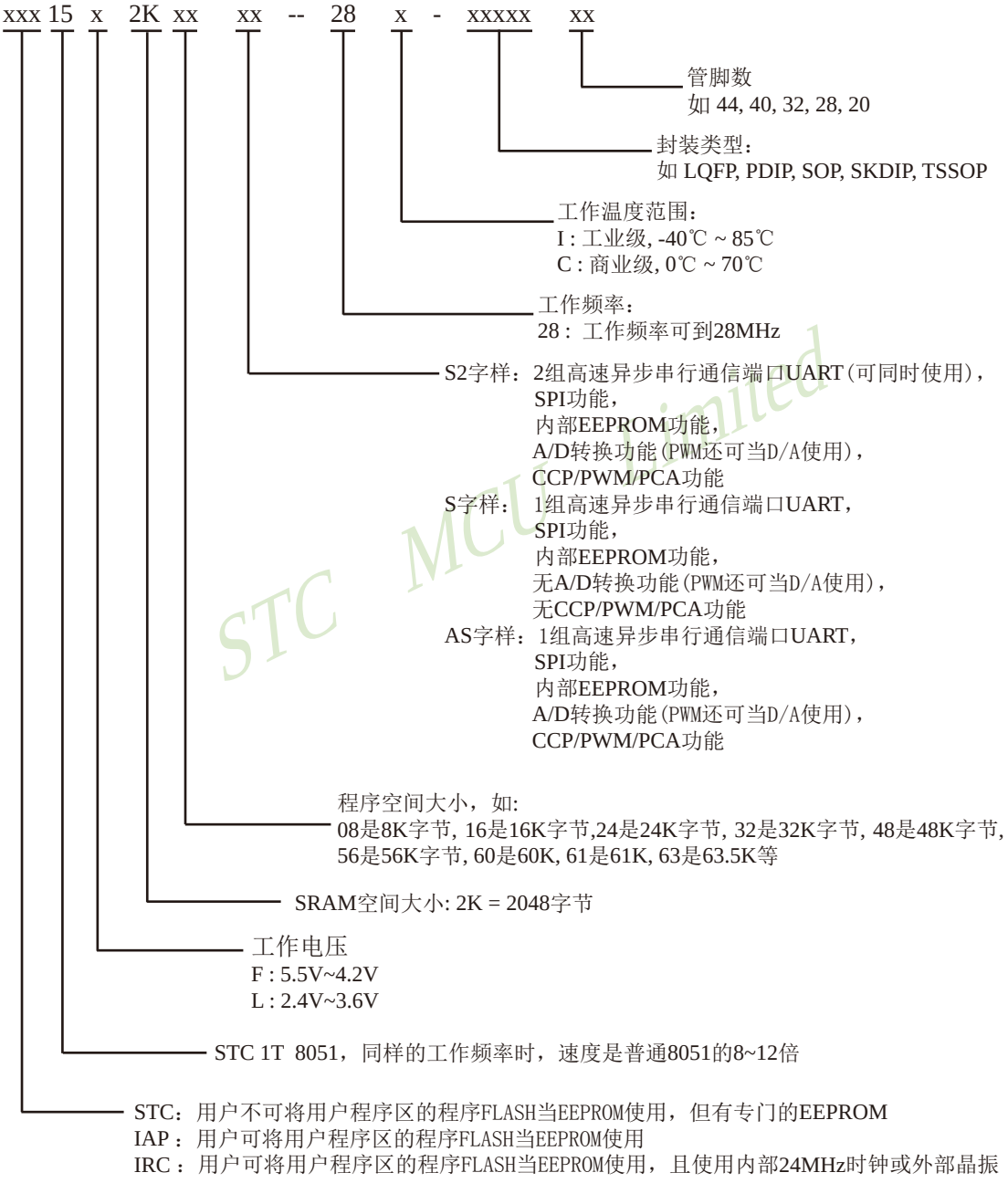
提供客制化IC服务

以上单价为10K起订
量小每片需加0.1元
以上价格运费由客户承担,零售10片起
如对价格不满，可来电要求降价

STC15F2K24AS-LQFP44,由于是无利润产品,不零售,100K起定,小量建议用STC152K08S2系列或STC15W1K16S或STC15W408AS系列取代

如果用户要用40-pin及以上的单片机，建议选用LQFP44的封装，但PDIP40封装仍正常供货；如果用户要用32-pin单片机，建议用户选用LQFP32封装；如果用户要用28-pin单片机，建议用户选用SOP28封装。

1.1.6 STC15F2K60S2系列单片机命名规则



STC MCU Limited

命名举例：

（1）STC15F2K60S2 - 28I - LQFP44 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~4.2V，SRAM空间大小为2K(2048)字节，程序空间大小为60K，有两组高速异步串行通信端口UART及1组SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

（2）STC15L2K60S2 - 28I - LQFP44 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为2.4V~3.6V，SRAM空间大小为2K(2048)字节，程序空间大小为60K，有两组高速异步串行通信端口UART及1组SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

（3）IAP15F2K61S2 - 28I - LQFP44 表示：
用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~4.2V，SRAM空间大小为2K(2048)字节，程序空间大小为61K，有两组高速异步串行通信端口UART及1组SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

（4）IAP15L2K61S2 - 28I - LQFP44 表示：
用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为2.4V~3.6V，SRAM空间大小为2K(2048)字节，程序空间大小为61K，有两组高速异步串行通信端口UART及1组SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

（5）IAP15F2K61S - 28I - LQFP44 表示：
用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~4.2V，SRAM空间大小为2K(2048)字节，程序空间大小为61K，有1组高速异步串行通信端口UART及1组SPI、内部EEPROM、工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

（6）IAP15L2K61S - 28I - LQFP44 表示：
用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为2.4V~3.6V，SRAM空间大小为2K(2048)字节，程序空间大小为61K，，有1组高速异步串行通信端口UART及1组SPI、内部EEPROM、工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

STC GXWMCU Limited

（7）IRC15F2K63S2 - 28I - LQFP44 表示：

用户可以将用户程序区的程序FLASH当EEPROM使用，且使用内部24MHz时钟或外部晶振，内部该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~4.2V，SRAM空间大小为2K(2048)字节，程序空间大小为63.5K，有两组高速异步串行通信端口UART及1组SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

（8）STC15F2K24AS - 28I - LQFP44 表示：

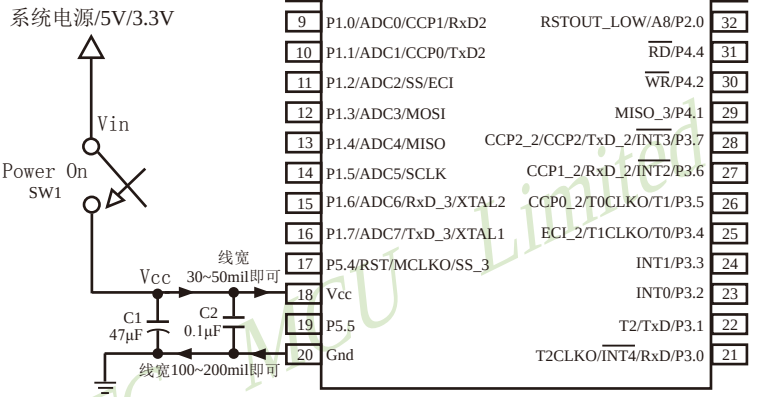
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~4.2V，SRAM空间大小为2K(2048)字节，程序空间大小为24K，有1组高速异步串行通信端口UART及1组SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

STC MCU Limited

1.1.7 STC15F2K60S2系列单片机最小应用系统

特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。



内部高可靠复位，可彻底省掉外部复位电路
P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

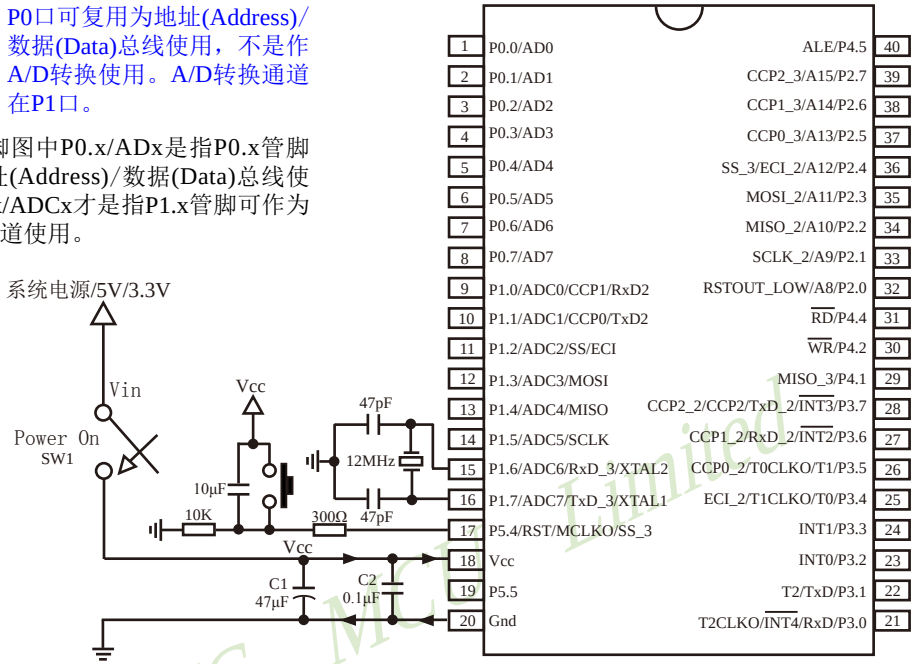
内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.1.8 STC15F2K60S2系列接外部晶振和外部复位(实际不需要)的线路图

特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。



内部高可靠复位，可彻底省掉外部复位电路，当然也可以使用外部复位电路，如上图所示。

P5.4/RST/MCLKO脚出厂时默认为I/O口，如需做RST复位脚，须在 STC-ISP 编程烧录时将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振，当然也可以使用外部晶振，如上图所示。单片机默认使用内部高精度R/C时钟，如用户需使用外部晶振，则须在STC-ISP 编程烧录时选择“外部晶体或时钟”。

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

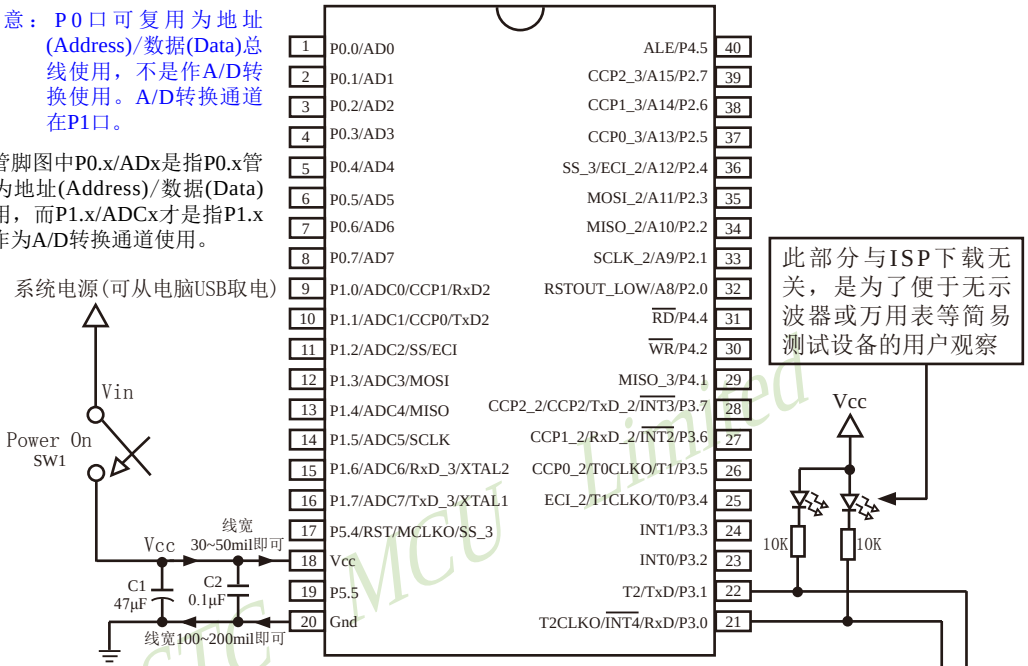
STC MCU Limited

1.1.9 STC15F2K60S2系列在系统可编程(ISP)及仿真典型应用线路图

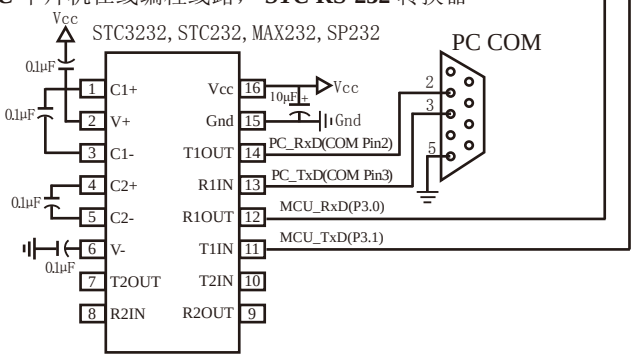
1.1.9.1 利用RS-232转换器的ISP下载编程典型应用线路图及仿真典型应用线路图

特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。



STC 单片机在线编程线路， ***STC*** **RS-232** 转换器



注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口1放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

内部高可靠复位，可彻底省掉外部复位电路，当然也可以使用外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

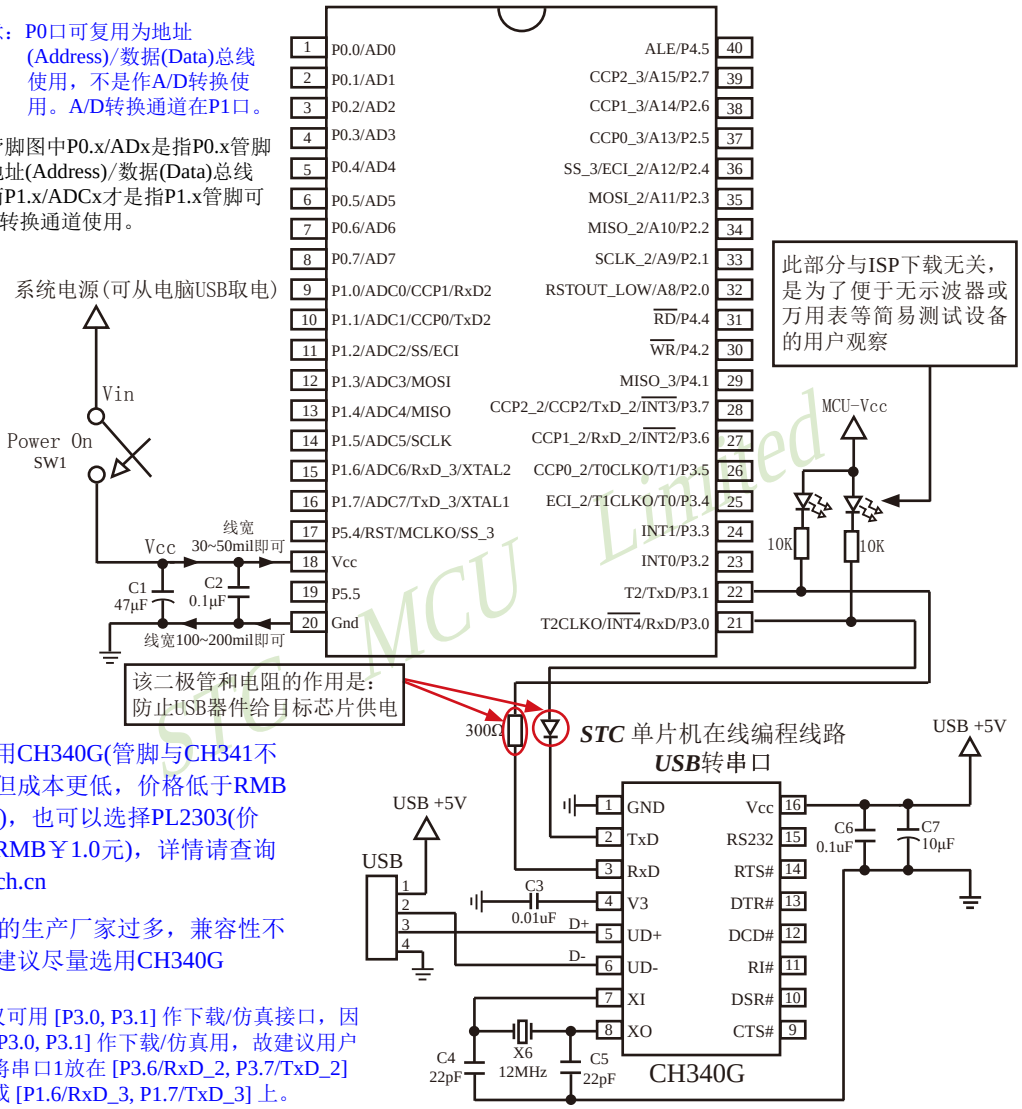
内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振，当然也可以使用外部晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.1.9.2 利用USB转串口的ISP下载编程典型应用线路图及仿真典型应用线路图

特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口1放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.1.10 STC15F2K60S2系列管脚说明

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	TSSOP20		
P0.0/AD0	40	2	1	1	29	-		P0：P0口既可作为输入/输出口，也可作为地址/数据复用总线使用。当P0口作为输入/输出口时，P0可以由软件配置成准双向口/弱上拉、推挽输出/强上拉、仅为输入(高阻)及开漏输出等4种工作类型之一，上电复位后为准双向口/弱上拉模式。当P0作为地址/数据复用总线使用时，是低8位地址线[A0~A7]及数据线的[D0~D7]。	
P0.1/AD1	41	3	2	2	30	-			
P0.2/AD2	42	4	3	3	31	-			
P0.3/AD3	43	5	4	4	32	-			
P0.4/AD4	44	6	5	-	-	-			
P0.5/AD5	1	7	6	-	-	-			
P0.6/AD5	2	8	7	-	-	-			
P0.7/AD7	3	9	8	-	-	-			
P1.0/ ADC0/ CCP1/ RxD2	4	10	9	5	1	3	1	P1.0	标准I/O口 PORT1[0]
								ADC0	ADC 输入通道-0
								CCP1	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-1
								RxD2	串口2数据接收端
P1.1/ ADC1/ CCP0/ TxD2	5	11	10	6	2	4	2	P1.1	标准I/O口 PORT1[1]
								ADC1	ADC 输入通道-1
								CCP0	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-0
								TxD2	串口2数据发送端
P1.2/ ADC2/ SS/ECI	7	13	11	7	3	5	20	P1.2	标准I/O口 PORT1[2]
								ADC2	ADC 输入通道-2
								SS	SPI同步串行接口的从机选择信号
								ECI	CCP / PCA计数器的外部脉冲输入脚
P1.3/ ADC3/ MOSI	8	14	12	8	4	6	19	P1.3	标准I/O口 PORT1[3]
								ADC3	ADC 输入通道-3
								MOSI	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P1.4/ ADC4/ MISO	9	15	13	9	5	7	3	P1.4	标准I/O口 PORT1[4]
								ADC4	ADC 输入通道-4
								MISO	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P1.5/ ADC5/ SCLK	10	16	14	10	6	8	4	P1.5	标准I/O口 PORT1[5]
								ADC5	ADC 输入通道-5
								SCLK	SPI同步串行接口的时钟信号

STC MCU Limited

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	TSSOP20		
P1.6/ADC6/ RxD_3/ XTAL2	11	17	15	11	7	9	5	P1.6	标准I/O口 PORT1[6]
								ADC6	ADC 输入通道-6
								RxD_3	串口1数据接收端
								XTAL2	内部时钟电路反相放大器的输出端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚可浮空，此时XTAL2实际将XTAL1输入的时钟进行输出。
P1.7/ADC7/ TxD_3/ XTAL1	12	18	16	12	8	10	6	P1.7	标准I/O口 PORT1[7]
								ADC7	ADC 输入通道-7
								TxD_3	串口1数据发送端
								XTAL1	内部时钟电路反相放大器输入端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚是外部时钟源的输入端。
P2.0/A8/ RSTOUT_LOW	30	36	32	25	21	23		P2.0	标准I/O口 PORT2[0]
								A8	地址总线第8位 — A8
								RSTOUT_LOW	上电后,输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P2.1/A9/ SCLK_2	31	37	33	26	22	24		P2.1	标准I/O口 PORT2[1]
								A9	地址总线第9位 — A9
								SCLK_2	SPI同步串行接口的时钟信号
P2.2/A10/ MISO_2	32	38	34	27	23	25		P2.2	标准I/O口 PORT2[2]
								A10	地址总线第10位 — A10
								MISO_2	SPI同步串行接口的主入从出（主器件的输入和从器件的输出）
P2.3/A11/ MOSI_2	33	39	35	28	24	26		P2.3	标准I/O口 PORT2[3]
								A11	地址总线第11位 — A11
								MOSI_2	SPI同步串行接口的主出从入（主器件的输出和从器件的输入）
P2.4/A12/ ECI_3/SS_2	34	40	36	29	25	27		P2.4	标准I/O口 PORT2[4]
								A12	地址总线第12位 — A12
								ECI_3	CCP / PCA计数器的外部脉冲输入脚
								SS_2	SPI同步串行接口的从机选择信号

STC MCU Limited

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	TSSOP20		
P2.5/A13/ CCP0_3	35	41	37	30	26	28		P2.5	标准I/O口 PORT2[5]
								A13	地址总线第13位 — A13
								CCP0_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P2.6/A14/ CCP1_3	36	42	38	31	27	1		P2.6	标准I/O口 PORT2[6]
								A14	地址总线第14位 — A14
								CCP1_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
P2.7/A15/ CCP2_3	37	43	39	32	28	2		P2.7	标准I/O口 PORT2[7]
								A15	地址总线第15位 — A15
								CCP2_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
P3.0/ RxD/ $\overline{\text{INT4}}$ /T2CLKO	18	24	21	17	13	15	11	P3.0	标准I/O口 PORT3[0]
								RxD	串口1数据接收端
								$\overline{\text{INT4}}$	外部中断4,只能下降沿中断,/INT4支持掉电唤醒
								T2CLKO	T2的时钟输出 可通过设置INT_CLKO[2]位/T2CLKO将该管脚配置为T2CLKO
P3.1/ TxD/T2	19	25	22	18	14	16	12	P3.1	标准I/O口 PORT3[1]
								TxD	串口1数据发送端
								T2	定时器/计数器2的外部输入
P3.2/ INT0	20	26	23	19	15	17	13	P3.2	标准I/O口 PORT3[2]
								INT0	外部中断0,既可上升沿中断也可下降沿中断. 如果IT0(TCON.0)被置为1,INT0管脚仅为下降沿中断。如果IT0(TCON.0)被清0，INT0管脚既支持上升沿中断也支持下降沿中断。 INT0支持掉电唤醒。
P3.3/ INT1	21	27	24	20	16	18	18	P3.3	标准I/O口 PORT3[3]
								INT1	外部中断1,既可上升沿中断也可下降沿中断. 如果IT1(TCON.2)被置为1,INT1管脚仅为下降沿中断。如果IT1(TCON.2)被清0，INT1管脚既支持上升沿中断也支持下降沿中断。 INT1支持掉电唤醒。

STC MCU Limited

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	TSSOP20		
P3.4/T0/ T1CLKO/ ECI_2	22	28	25	21	17	19	14	P3.4	标准I/O口 PORT3[4]
								T0	定时器/计数器0的外部输入
								T1CLKO	定时器/计数器1的时钟输出 可通过设置INT_CLKO[1]位 /T1CLKO将该管脚配置为 T1CLKO，也可对T1脚的外部时钟输入进行分频输出
								ECI_2	CCP / PCA计数器的外部脉冲输入脚
P3.5/T1/ T0CLKO/ CCP0_2	23	29	26	22	18	20	15	P3.5	标准I/O口 PORT3[5]
								T1	定时器/计数器1的外部输入
								T0CLKO	定时器/计数器0的时钟输出 可通过设置INT_CLKO[0]位 /T0CLKO将该管脚配置为 T0CLKO，也可对T0脚的外部时钟输入进行分频输出
								CCP0_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P3.6/$\overline{\text{INT2}}$ /RxD_2/ CCP1_2	24	30	27	23	19	21	16	P3.6	标准I/O口 PORT3[6]
								$\overline{\text{INT2}}$	外部中断2, 只能下降沿中断 支持掉电唤醒
								RxD_2	串口1数据接收端
								CCP1_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
P3.7/$\overline{\text{INT3}}$ /TxD_2/ CCP2/ CCP2_2	25	31	28	24	20	22	17	P3.7	标准I/O口 PORT3[7]
								$\overline{\text{INT3}}$	外部中断3, 只能下降沿中断 支持掉电唤醒
								TxD_2	串口1数据发送端
								CCP2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
								CCP2_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
P4.0/ MOSI_3	17	23	-	-	-	-		P4.0	标准I/O口 PORT4[0]
								MISO_3	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P4.1/ MISO_3	26	32	29	-	-	-		P4.1	标准I/O口 PORT4[1]
								MOSI_3	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)

STC MCU Limited

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	TSSOP20		
P4.2/$\overline{WR}$	27	33	30	-	-	-		P4.2	标准I/O口 PORT4[2]
								$\overline{WR}$	外部数据存储器写脉冲
P4.3/SCLK_3	28	34	-	-	-	-		P4.3	标准I/O口 PORT4[3]
								SCLK_3	SPI同步串行接口的时钟信号
P4.4/$\overline{RD}$	29	35	31	-	-	-		P4.4	标准I/O口 PORT4[4]
								$\overline{RD}$	外部数据存储器读脉冲
P4.5/ALE	38	44	40	-	-	-		P4.5	标准I/O口 PORT4[5]
								ALE	地址锁存允许
P4.6/RxD2_2	39	1	-	-	-	-		P4.6	标准I/O口 PORT4[6]
								RxD2_2	串口2数据接收端
P4.7/TxD2_2	6	12	-	-	-	-		P4.7	标准I/O口 PORT4[7]
								TxD2_2	串口2数据发送端
P5.4/RST/ MCLKO/ SS_3	13	19	17	13	9	11	7	P5.4	标准I/O口 PORT5[4]
								RST	复位脚(高电平复位)
								MCLKO	主时钟输出：输出的频率可为MCLK/1, MCLK/2, MCLK/4（MCLK是指主时钟频率）。现供货的STC15F2K60S2系列C版本单片机的主时钟只可以是内部R/C时钟，STC15F2K60S2系列下一升级版本——STC15W2K60S2系列单片机主时钟既可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。
								SS_3	SPI同步串行接口的从机选择信号
P5.5	15	21	19	15	11	13	9	标准I/O口 PORT5[5]	
Vcc	14	20	18	14	10	12	8	电源正极	
Gnd	16	22	20	16	12	14	10	电源负极，接地	

STC MCU Limited

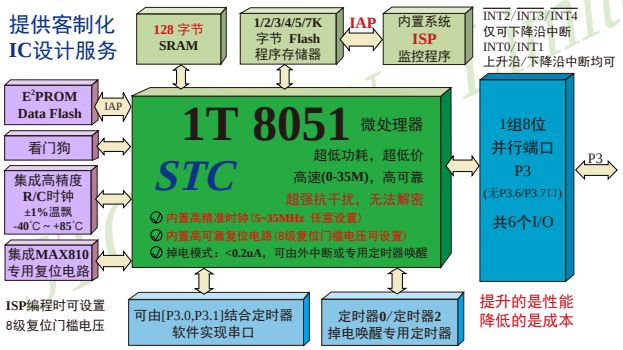
1.2 STC15F101W系列单片机总体介绍(B版大批量现货供应中)
——STC15L101W系列建议用STC15W104系列取代

1.2.1 STC15F101W系列单片机简介(与STC15F104E系列有所不兼容)

STC15F101W系列单片机是STC生产的单时钟/机器周期(1T)的单片机，是高速/高可靠/低功耗/超强抗干扰的新一代8051单片机，采用STC第八代加密技术，无法解密，指令代码完全兼容传统8051,但速度快8-12倍。内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP编程时8级复位门槛电压可选)。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通8051快8-12倍
2. 工作电压：
   STC15F101W 系列工作电压：5.5V - 3.8V（5V 单片机）
   STC15L101W 系列工作电压：3.6V - 2.4V（3V 单片机）
3. 1K/2K/3K/4K/5K/7K字节片内Flash程序存储器，可擦写次数10万次以上
4. 片内128字节的SRAM
5. 有片内EEPROM功能，擦写次数10万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 内部高可靠复位，ISP编程时8级复位门槛电压可选，可彻底省掉外部复位电路
8. 内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时内部时钟从5MHz~35MHz可设(5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz)
9. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号

10.工作频率范围：5MHz ~ 35MHz，相当于普通8051的60MHz～420MHz

11.串口功能可由[P3.0/$\overline{\text{INT4}}$, P3.1]结合定时器实现

12. 支持程序加密后传输，防拦截

13.支持RS485下载

14.低功耗设计：低速模式，空闲模式，掉电模式/停机模式.

15.可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。

16.可将掉电模式/停机模式唤醒的资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.4, $\overline{\text{INT3}}$/P3.5, $\overline{\text{INT4}}$/P3.0($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚T0/T2(下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；内部低功耗掉电唤醒专用定时器。

17. 共2个定时器/计数器——T0(兼容普通8051的定时器)/T2，并均可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4）

18. 可编程时钟输出功能(对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出)：
由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz.；
而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz.

① T0在P3.5/T0CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；

② T2在P3.0/T2CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；

以上2个定时器/计数器均可1~65536级分频输出。

③ 主时钟在P3.4/MCLKO对外输出时钟，并可如下分频MCLK/1, MCLK/2, MCLK/4.
STC15F101W系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。
MCLK是指主时钟频率，MCLKO是指主时钟输出。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/P1.6口对外输出主时钟。

19. 硬件看门狗(WDT)

20. 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令

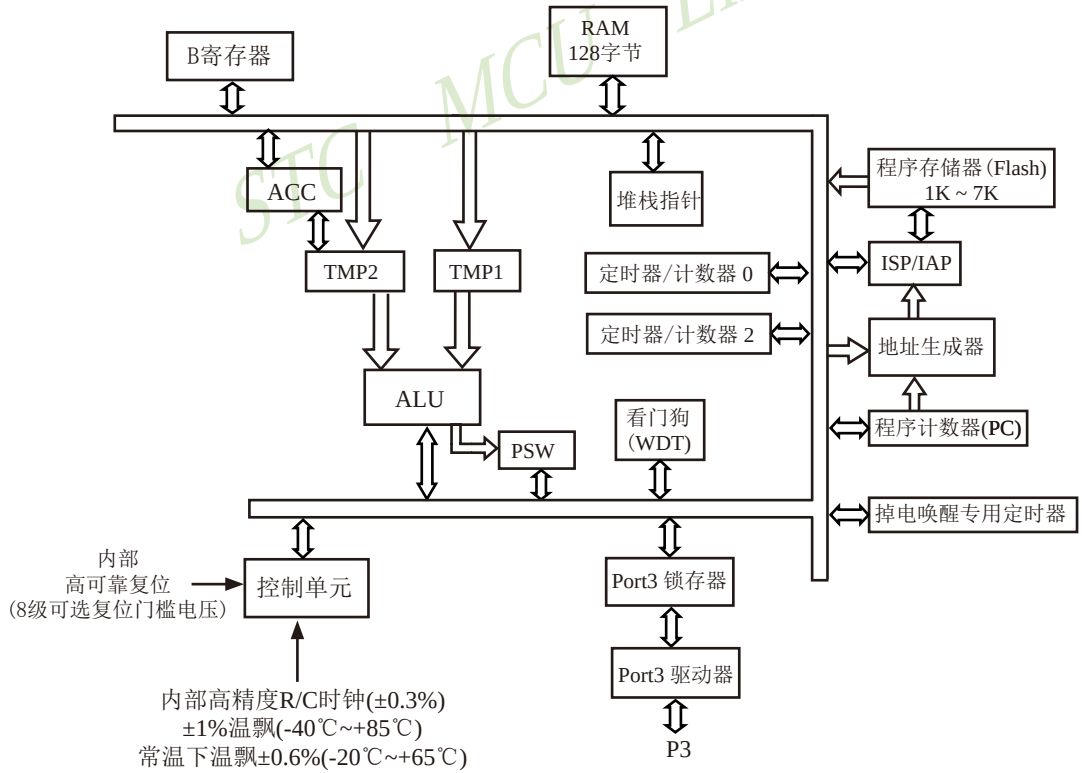
21. 共6个通用I/O口，复位后为： 准双向口/弱上拉（普通8051传统I/O口）
可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏

每个I/O口驱动能力均可达到20mA，但整个芯片电流最大不要超过90mA.
如果I/O口不够用，可外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口。

22. 封装：SOP-8, DIP-8, DFN-8(不推荐).
23. 全部175℃八小时高温烘烤，高品质制造保证
24. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

1.2.2 STC15F101W系列单片机的内部结构图

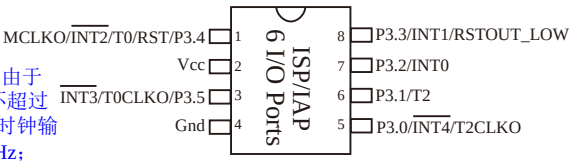
STC15F101W系列单片机的内部结构框图如下图所示。STC15F101W系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、掉电唤醒专用定时器、I/O口、看门狗、片内高精度R/C时钟及高可靠复位等模块。



STC15F101W系列内部结构框图

1.2.3 STC15F101W系列单片机管脚图

所有封装形式均满足欧盟RoHS要求，强烈推荐选择SOP-8贴片封装，传统的插件DIP-8封装稳定供货。



SOP-8/DIP-8
DFN-8

MCLKO是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/**1, MCLK/2, MCLK/4.**

此系列的主时钟只可以是内部R/C时钟。MCLK是指主时钟频率。

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；

T0CLKO是指定时器/计数器0的可编程时钟输出(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；

T2CLKO是指定时器/计数器2的可编程时钟输出(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；

T0CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T2的时钟输入进行时钟分频输出，作分频器使用。

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	00x0,x000

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， **输出时钟频率** = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， **输出时钟频率** = MCLK / 4

STC15F101W系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟，MCLK是指主时钟频率。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列)均在MCLKO/P5.4口对外输出主时钟。

Tx_Rx：P3.1口的对外输出实时反映P3.0口的外部输入状态的选择位

0：P3.1口的对外输出不反映P3.0口的外部输入状态

1：将P3.0管脚输入的电平状态实时输出在P3.1外部管脚上，即P3.1口的对外输出实时反映P3.0口的外部输入状态。当P3.0外部输入为1时，P3.1口的对外输出就为1；当P3.0外部输入为0时，P3.1口的对外输出也就为0。

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给CPU、定时器的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

STC15F101W系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。

1.2.4 STC15F101W系列单片机选型价格一览表

型号	工作电压（V）	Flash程序存储器(字节byte)	SRAM字节	串行口并可掉电唤醒	SPI	定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	8-Pin封装 SOP8 / DIP8 / DFN8 (6个I/O口) 价格(RMB ¥)		
																				SOP8	DIP8	DFN8
STC15F101W系列单片机选型价格一览表																						
STC15F100W	5.5-3.8	0.5K	128	-	-	2	-	有	5	-	1	-	有	有	8级	有	是	有	是	¥0.99		
STC15F101W	5.5-3.8	1K	128	-	-	2	-	有	5	-	1	4K	有	有	8级	有	是	有	是	¥1.2		
STC15F102W	5.5-3.8	2K	128	-	-	2	-	有	5	-	1	3K	有	有	8级	有	是	有	是	¥1.3	¥1.5	
STC15F103W	5.5-3.8	3K	128	-	-	2	-	有	5	–	1	2K	有	有	8级	有	是	有	是	¥1.3	¥1.5	
STC15F104W	5.5-3.8	4K	128	-	-	2	-	有	5	–	1	1K	有	有	8级	有	是	有	是	¥1.4	¥1.6	¥1.5
IAP15F105W	5.5-3.8	5K	128	-	-	2	-	有	5	–	1	IAP	有	有	8级	有	是	有	是	¥1.4	¥1.6	
																				用户可将用户程序区的程序FLASH当EEPROM使用		
IRC15F107W 固定使用内部24MHz时钟	5.5-3.8	7K	128	-	-	2	-	有	5	–	1	IAP	有	有	固定	有	是	无	否	¥1.4	¥1.6	
																				用户可将用户程序区的程序FLASH当EEPROM使用		
STC15L101W系列单片机选型价格一览表，此系列建议用STC15W104系列取代（STC15W104系列大批量现货供应中）																						
STC15L100W	3.6-2.4	0.5K	128	-	-	2	-	有	5	-	1	-	有	有	8级	有	是	有	是	¥0.99		
STC15L101W	3.6-2.4	1K	128	-	-	2	-	有	5	-	1	4K	有	有	8级	有	是	有	是	¥1.2		
STC15L102W	3.6-2.4	2K	128	-	-	2	-	有	5	–	1	3K	有	有	8级	有	是	有	是	¥1.3		
STC15L104W	3.6-2.4	4K	128	-	-	2	-	有	5	–	1	1K	有	有	8级	有	是	有	是	¥1.4		
IAP15L105W	3.6-2.4	5K	128	-	-	2	-	有	5	–	1	IAP	有	有	8级	有	是	有	是	¥1.4	¥1.6	
																				用户可将用户程序区的程序FLASH当EEPROM使用		

STC MCU Limited

提供客制化IC服务

建议用户选用SOP8封装，但DIP8封装以及新生产DFN8封装仍正常供货。

STC15F100W和STC15L100W由于是无利润产品,不零售,200K起定,
小量建议用STC15F101W和STC15L101W取代

以上单价为10K起订
量小每片需加0.1元
以上价格运费由客户承担,零售10片起
如对价格不满，可来电要求降价

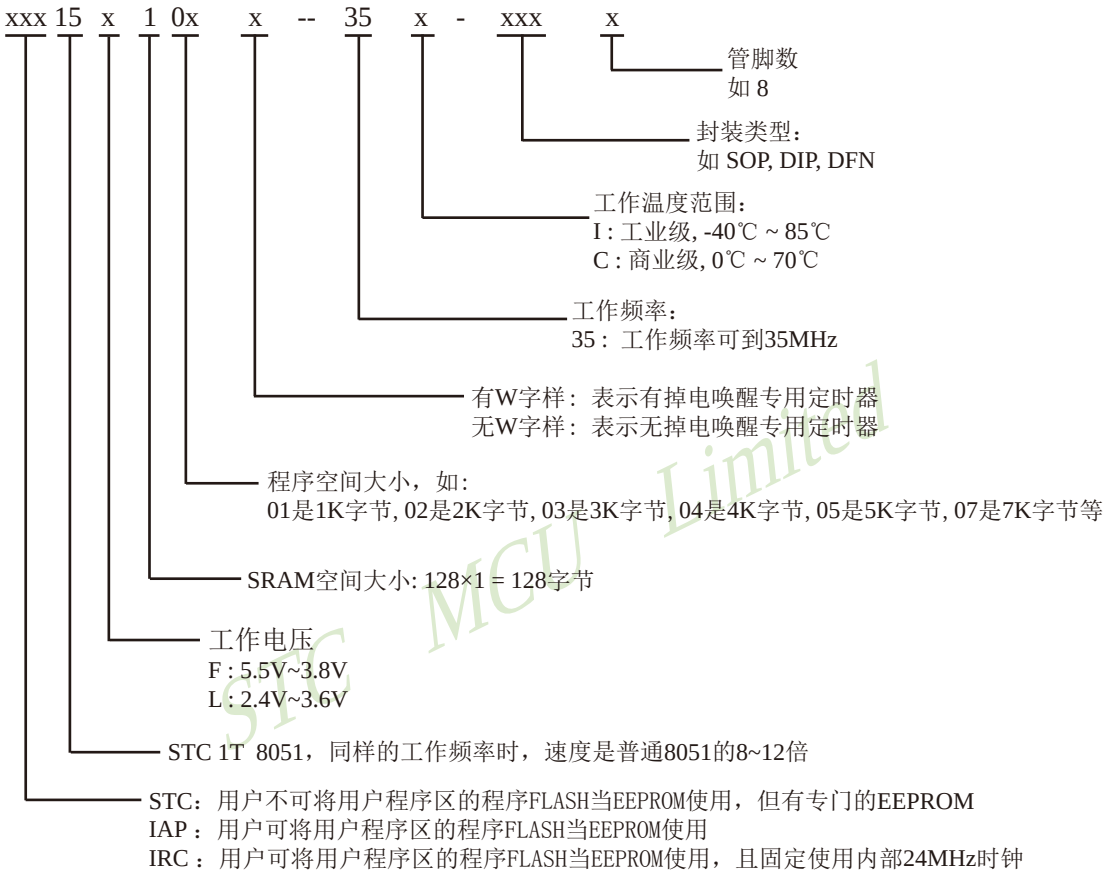
程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

因为程序区的最后7个字节单元被强制性的放入全球唯一ID号的内容,所以用户实际可以使用的程序空间大小要比选型表中的大小少7个字节

特别声明：以15L开头的芯片如需进入"掉电模式",进入"掉电模式"前必须启动掉电唤醒定时器<3uA>,不超过1秒要唤醒一次，以15F和15W开头的芯片则不需要

总结：STC15F/L101W系列单片机(含IRC15F107W型号单片机)有两个16位重装载定时器/计数器(这两个定时器/计数器分别是：定时器/计数器0和定时器/计数器2)，有5个外部中断INT0/INT1/INT2/INT3/INT4，有掉电唤醒专用定时器，有1个数据指针DPTR。表中“-”表示该型号的单片机无相应的功能。STC15F/L101W系列单片机(含IRC15F107W型号单片机)无串行口、无SPI、无A/D转换、无CCP/PWM/PCA、无外部数据总线等功能。

1.2.5 STC15F101W系列单片机命名规则



命名举例：

（1）STC15F101W- 35I - SOP8 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~3.8V，SRAM空间大小为128字节，程序空间大小为1K，有掉电唤醒专用定时器，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为8。

（2）STC15L101W- 35I - SOP8 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为2.4V~3.6V，SRAM空间大小为128字节，程序空间大小为1K，有掉电唤醒专用定时器，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为8。

STC MCU Limited

（3）STC15F104W- 35I - SOP8 表示：

用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~3.8V，SRAM空间大小为128字节，程序空间大小为4K，有掉电唤醒专用定时器，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为8。

（4）STC15L104W- 35I - SOP8 表示：

用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为2.4V~3.6V，SRAM空间大小为128字节，程序空间大小为4K，有掉电唤醒专用定时器，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为8。

（5）IAP15F105W- 35I - SOP8 表示：

用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~3.8V，SRAM空间大小为128字节，程序空间大小为5K，有掉电唤醒专用定时器，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为8。

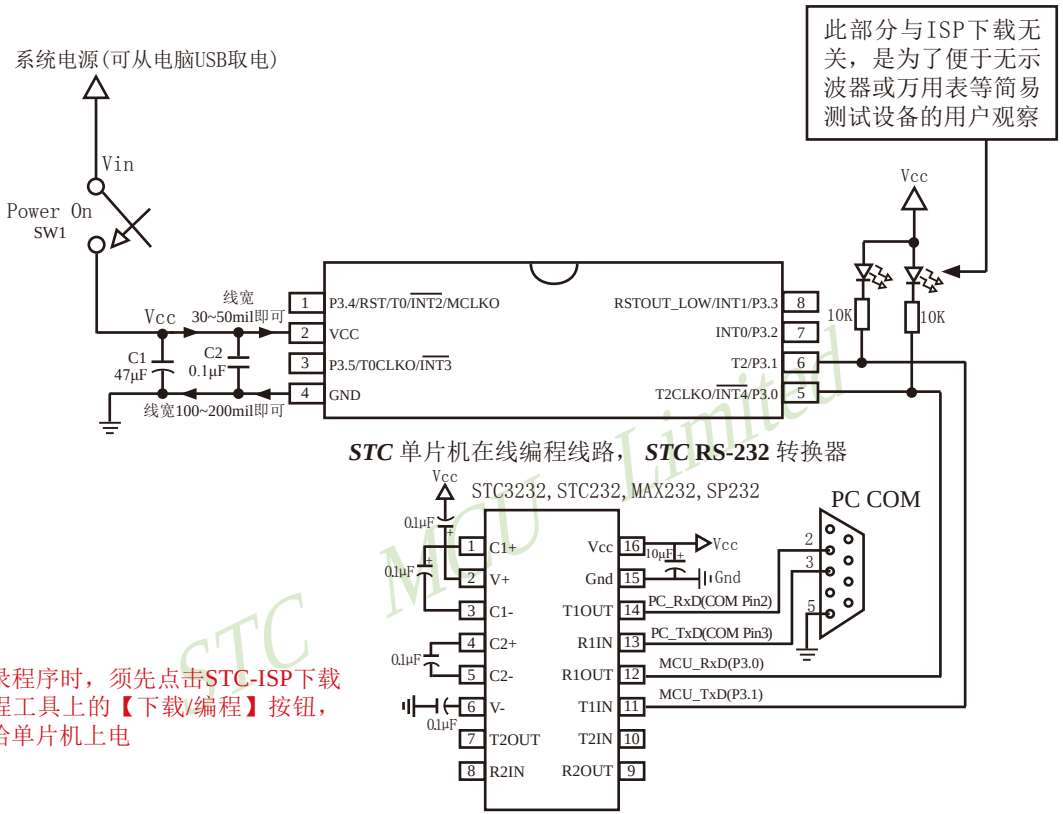
（6）IAP15L105W- 35I - SOP8 表示：

用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为2.4V~3.6V，SRAM空间大小为128字节，程序空间大小为5K，有掉电唤醒专用定时器，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为8。

STC MCU Limited

1.2.6 STC15F101W系列单片机在系统可编程(ISP)典型应用线路图

1.2.6.1 利用RS-232转换器的ISP下载编程典型应用线路图



烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

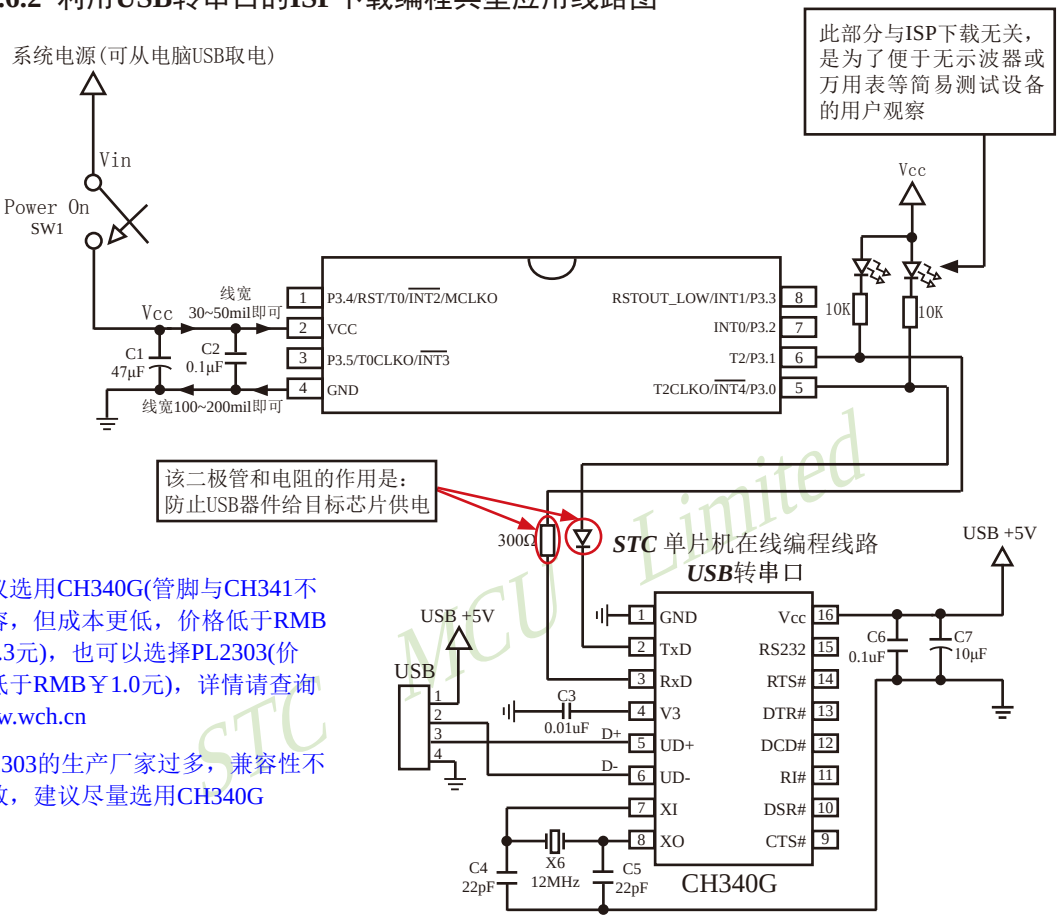
内部高可靠复位，可彻底省掉外部复位电路

P3.4/RST/T0/$\overline{\text{INT2}}$/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.2.6.2 利用USB转串口的ISP下载编程典型应用线路图



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

内部高可靠复位，可彻底省掉外部复位电路

P3.4/RST/T0/$\overline{\text{INT2}}$/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.2.7 STC15F101W系列单片机的管脚说明

管脚	管脚编号 (封装SOP-8/DIP-8/ DFN-8)	说明	
P3.0/$\overline{\text{INT4}}$ /T2CLKO	5	P3.0	标准I/O口 PORT3[0]
		$\overline{\text{INT4}}$	外部中断4, 只能下降沿中断 $\overline{\text{INT4}}$支持掉电唤醒
		T2CLKO	T2的时钟输出 可通过设置INT_CLKO[2]位/T2CLKO将该管脚配置为T2CLKO
P3.1/T2	6	P3.1	标准I/O口 PORT3[1]
		T2	定时器/计数器2的外部输入
P3.2/INT0	7	P3.2	标准I/O口 PORT3[2]
		INT0	外部中断0，既可上升沿中断也可下降沿中断. 如果IT0(TCON.0)被置为1, INT0管脚仅为下降沿中断。如果IT0(TCON.0)被清0，INT0管脚既支持上升沿中断也支持下降沿中断。 INT0支持掉电唤醒。
P3.3/INT1/ RSTOUT_LOW	8	P3.3	标准I/O口 PORT3[3]
		INT1	外部中断1，既可上升沿中断也可下降沿中断. 如果IT1(TCON.2)被置为1, INT1管脚仅为下降沿中断。如果IT1(TCON.2)被清0，INT1管脚既支持上升沿中断也支持下降沿中断。 INT1支持掉电唤醒。
		RSTOUT_LOW	上电后, 输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P3.4/RST/T0/$\overline{\text{INT2}}$ /MCLKO	1	P3.4	标准I/O口 PORT3[4]
		RST	复位脚, 高电平复位
		T0	定时器/计数器0的外部输入
		$\overline{\text{INT2}}$	外部中断2, 只能下降沿中断 $\overline{\text{INT2}}$支持掉电唤醒
		MCLKO	主时钟输出;输出的频率可为MCLK/1, MCLK/2, MCLK/4（MCLK为主时钟频率）。 此系列的主时钟只可以是内部R/C时钟。
P3.5/T0CLKO/ $\overline{\text{INT3}}$	3	P3.5	标准I/O口 PORT3[5]
		T0CLKO	定时器/计数器0的时钟输出 可通过设置INT_CLKO[0]位/T0CLKO将该管脚配置为T0CLKO, 也可对T0脚的外部时钟输入进行分频输出
		$\overline{\text{INT3}}$	外部中断3, 只能下降沿中断 $\overline{\text{INT3}}$支持掉电唤醒
Vcc	2	电源正极	
Gnd	4	电源负极，接地	

STC MCU Limited

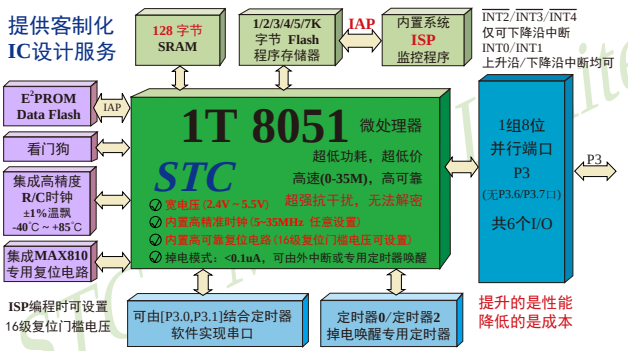
1.3 STC15W104系列单片机总体介绍(大批量现货供应中)

1.3.1 STC15W104系列单片机简介

STC15W104系列单片机是STC生产的单时钟/机器周期(1T)的单片机，是高速/高可靠/宽电压/低功耗/超强抗干扰的新一代8051单片机，采用STC第九代加密技术，无法解密，指令代码完全兼容传统8051,但速度快8-12倍。内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP编程时16级复位门槛电压可选)。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通8051快8-12倍
2. 工作电压：2.4V - 5.5V
3. 1K/2K/3K/4K/5K/7K字节片内Flash程序存储器，可擦写次数10万次以上
4. 片内128字节的SRAM
5. 有片内EEPROM功能，擦写次数10万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 内部高可靠复位，ISP编程时16级复位门槛电压可选，可彻底省掉外部复位电路
8. 内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时内部时钟从5MHz~35MHz可设(5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz)
9. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
10. 工作频率范围：5MHz ~ 35MHz，相当于普通8051的60MHz～420MHz
11. 串口功能可由[P3.0/$\overline{\text{INT4}}$,P3.1]结合定时器实现
12. 支持程序加密后传输，防拦截
13. 支持RS485下载

14.低功耗设计：低速模式，空闲模式，掉电模式/停机模式.

15.可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。

16.可将掉电模式/停机模式唤醒的资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{INT2}$/P3.4, $\overline{INT3}$/P3.5, $\overline{INT4}$/P3.0($\overline{INT2}$/$\overline{INT3}$/$\overline{INT4}$仅可下降沿中断)；管脚T0/T2(下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；内部低功耗掉电唤醒专用定时器。

17. 共2个定时器/计数器——T0(兼容普通8051的定时器)/T2，并均可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4）

18. 可编程时钟输出功能(对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出)：
由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz.;
而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz.
① T0在P3.5/T0CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出);
② T2在P3.0/T2CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出);
以上2个定时器/计数器均可1~65536级分频输出。
③ 主时钟在P3.4/MCLKO对外输出时钟，并可如下分频MCLK/1, MCLK/2, MCLK/4.
STC15W104系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。
MCLK是指主时钟频率，MCLKO是指主时钟输出。
STC15系列8-pin单片机(如STC15W104系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/P1.6口对外输出主时钟。

19. 硬件看门狗(WDT)

20. 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令

21. 共6个通用I/O口，复位后为： 准双向口/弱上拉（普通8051传统I/O口）
可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
每个I/O口驱动能力均可达到20mA，但整个芯片电流最大不要超过90mA.
如果I/O口不够用，可外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口。

22. 封装：SOP-8, DIP-8, DFN-8(不推荐).

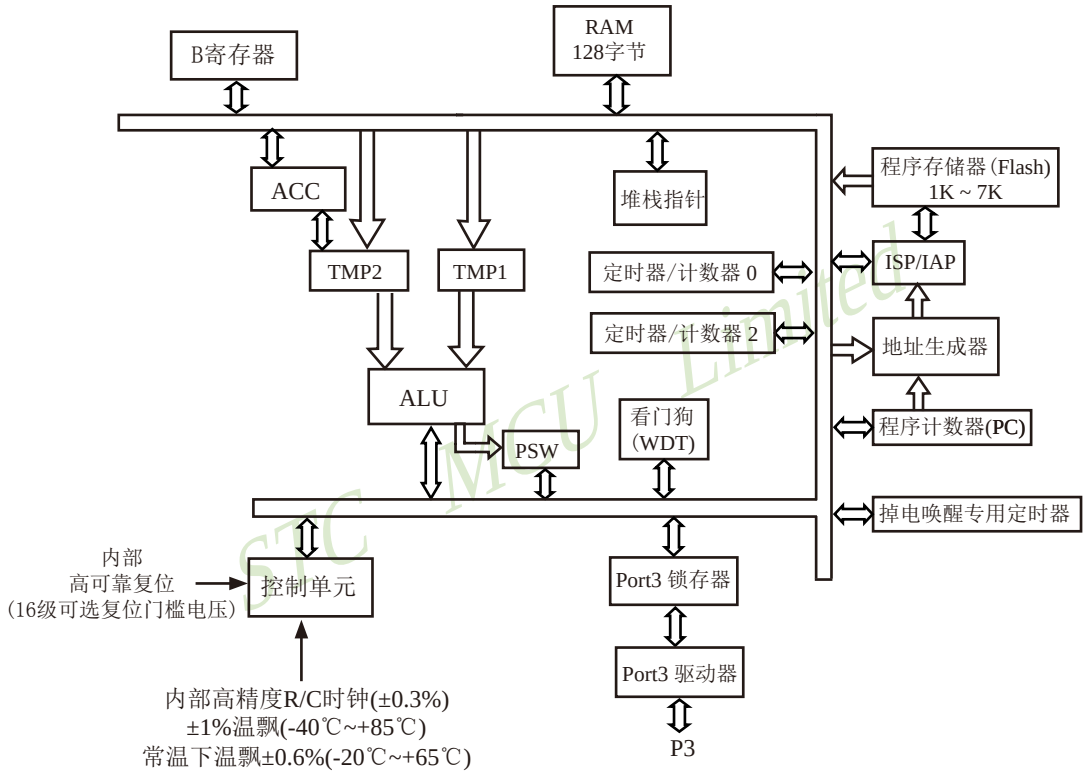
23. 全部175℃八小时高温烘烤，高品质制造保证

24. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

STC MCU Limited

1.3.2 STC15W104系列单片机的内部结构图

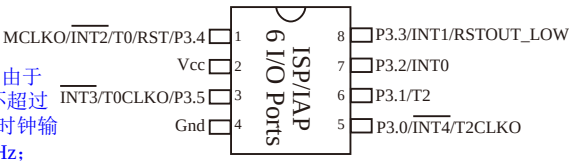
STC15W104系列单片机的内部结构框图如下图所示。STC15W104系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、掉电唤醒专用定时器、I/O口、看门狗、片内高精度R/C时钟及高可靠复位等模块。



STC15W104系列内部结构框图

1.3.3 STC15W104系列单片机管脚图

所有封装形式均满足欧盟RoHS要求，强烈推荐选择SOP-8贴片封装，传统的插件DIP-8封装稳定供货。



SOP-8/DIP-8
DFN-8

MCLKO是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/**1, MCLK/2, MCLK/4.**

此系列的主时钟只可以是内部R/C时钟。MCLK是指主时钟频率。

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；

T0CLKO是指定时器/计数器0的可编程时钟输出(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；

T2CLKO是指定时器/计数器2的可编程时钟输出(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；

T0CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T2的时钟输入进行时钟分频输出，作分频器使用。

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	00x0,x000

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， **输出时钟频率** = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， **输出时钟频率** = MCLK / 4

STC15W104系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟，MCLK是指主时钟频率。
STC15系列8-pin单片机(如STC15W104系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列)均在MCLKO/P5.4口对外输出主时钟。

Tx_Rx：P3.1口的对外输出实时反映P3.0口的外部输入状态的选择位
0：P3.1口的对外输出不反映P3.0口的外部输入状态
1：将P3.0管脚输入的电平状态实时输出在P3.1外部管脚上，即P3.1口的对外输出实时反映P3.0口的外部输入状态。当P3.0外部输入为1时，P3.1口的对外输出就为1；当P3.0外部输入为0时，P3.1口的对外输出也就为0。

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给CPU、定时器的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

STC15W104系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。

1.3.4 STC15W104系列单片机选型价格一览表

型号	工作电压（V）	Flash程序存储器(字节byte)	SRAM字节	串行口并可掉电唤醒	SPI	定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及低复位	程序加密后传输(防拦截)	支持RS485下载	8-Pin封装 SOP8 / DIP8 / DFN8 (6个I/O口) 价格(RMB ¥)		
																					SOP8	DIP8	DFN8
STC15W104系列单片机选型价格一览表																							
STC15W100	5.5-2.4	0.5K	128	-	-	2	-	有	5	-	-	1	-	有	有	16级	有	是	有	是	¥1.05		
STC15W101	5.5-2.4	1K	128	-	-	2	-	有	5	-	-	1	4K	有	有	16级	有	是	有	是	¥1.25		
STC15W102	5.5-2.4	2K	128	-	-	2	-	有	5	-	-	1	3K	有	有	16级	有	是	有	是	¥1.35	¥1.55	
STC15W103	5.5-2.4	3K	128	-	-	2	-	有	5	–	–	1	2K	有	有	16级	有	是	有	是	¥1.3	¥1.5	
STC15W104	5.5-2.4	4K	128	-	-	2	-	有	5	–	–	1	1K	有	有	16级	有	是	有	是	¥1.45	¥1.65	¥1.55
IAP15W105	5.5-2.4	5K	128	-	-	2	-	有	5	–	–	1	IAP	有	有	16级	有	是	有	是	¥1.45	¥1.65	
																					用户可将用户程序区的程序FLASH当EEPROM使用		
IRC15W107 固定使用内部24MHz时钟	5.5-2.4	7K	128	-	-	2	-	有	5	–	–	1	IAP	有	有	固定	有	是	无	否	¥1.45	¥1.65	
																					用户可将用户程序区的程序FLASH当EEPROM使用		

提供客制化IC服务

建议用户选用SOP8封装，但DIP8封装以及新生产DFN8封装仍正常供货。

STC15W100由于是无利润产品,不零售,200K起定,小批量建议用STC15W101取代

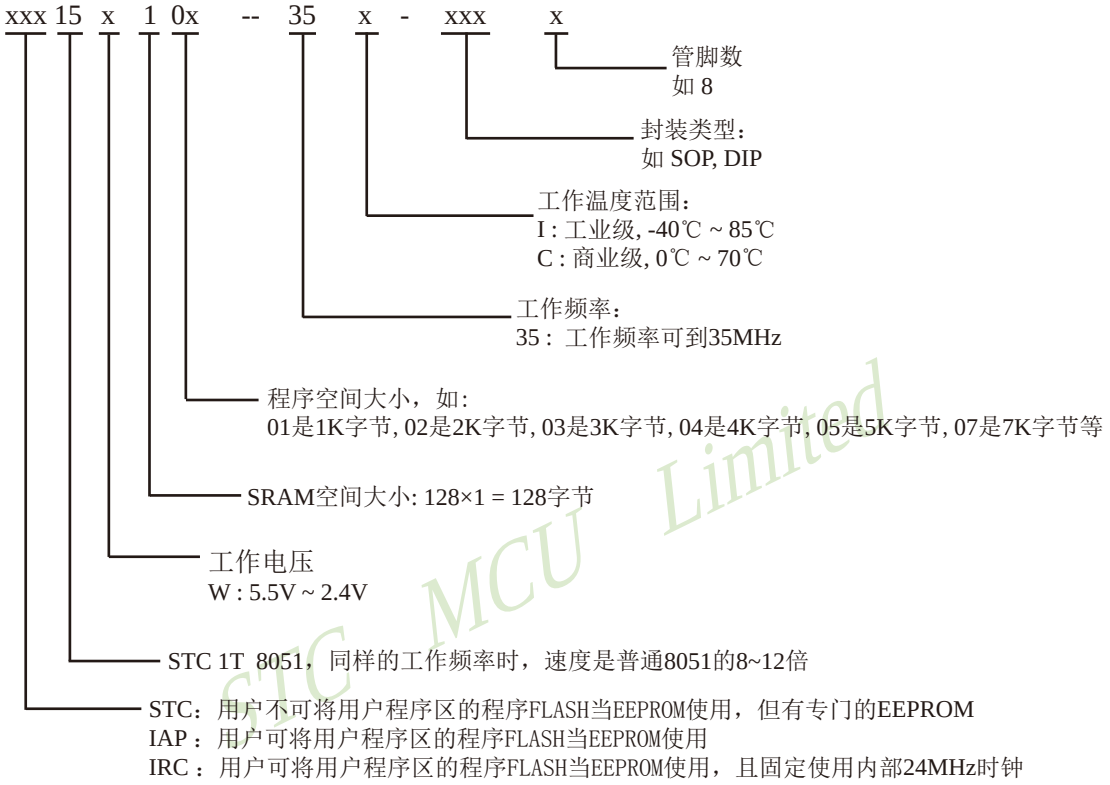
以上单价为10K起订
量小每片需加0.1元
以上价格运费由客户承担,零售10片起
如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

因为程序区的最后7个字节单元被强制性的放入全球唯一ID号的内容,所以用户实际可以使用的程序空间大小要比选型表中的大小少7个字节

总结：STC15W101系列单片机(含IRC15W107型号单片机)有两个16位重装载定时器/计数器(这两个定时器/计数器分别是：定时器/计数器0和定时器/计数器2)，有5个外部中断INT0/INT1/$\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$，有掉电唤醒专用定时器，有1个数据指针DPTR。表中"-"表示该型号的单片机无相应的功能。STC15F/L101W系列单片机(含IRC15F107W型号单片机)无串行口、无比较器、无SPI、无A/D转换、无CCP/PWM/PCA、无外部数据总线等功能。

1.2.5 STC15W104系列单片机命名规则



STC MCU Limited

命名举例：

（1）STC15W101- 35I - SOP8 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为128字节，程序空间大小为1K，有掉电唤醒专用定时器，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为8。

（2）STC15W104- 35I - SOP8 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为128字节，程序空间大小为4K，有掉电唤醒专用定时器，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为8。

（3）IAP15W105- 35I - SOP8 表示：

用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为128字节，程序空间大小为5K，有掉电唤醒专用定时器，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为8。

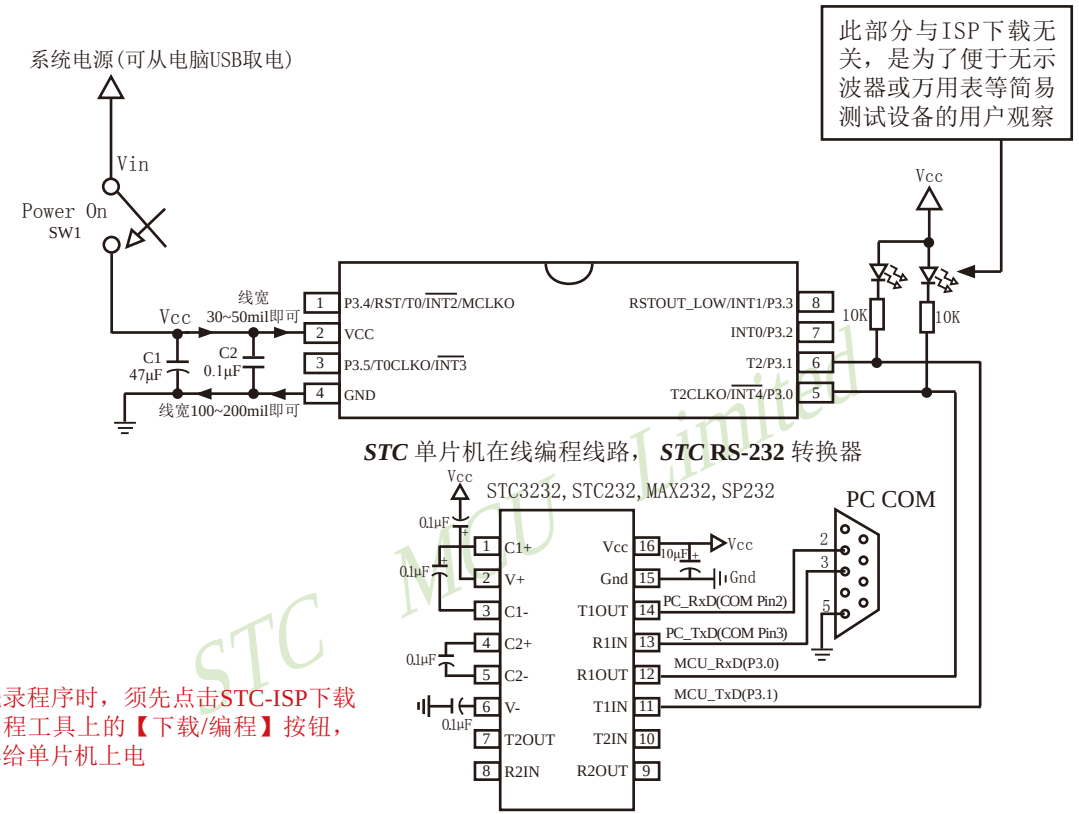
（4）IRC15W107- 35I - DIP8 表示：

用户可将用户程序区的程序FLASH当EEPROM使用，且固定使用内部24MHz时钟，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为128字节，程序空间大小为7K，有掉电唤醒专用定时器，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为DIP贴片封装，管脚数为8。

STC MCU Limited

1.3.6 STC15W104系列单片机在系统可编程(ISP)典型应用线路图

1.3.6.1 利用RS-232转换器的ISP下载编程典型应用线路图



烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

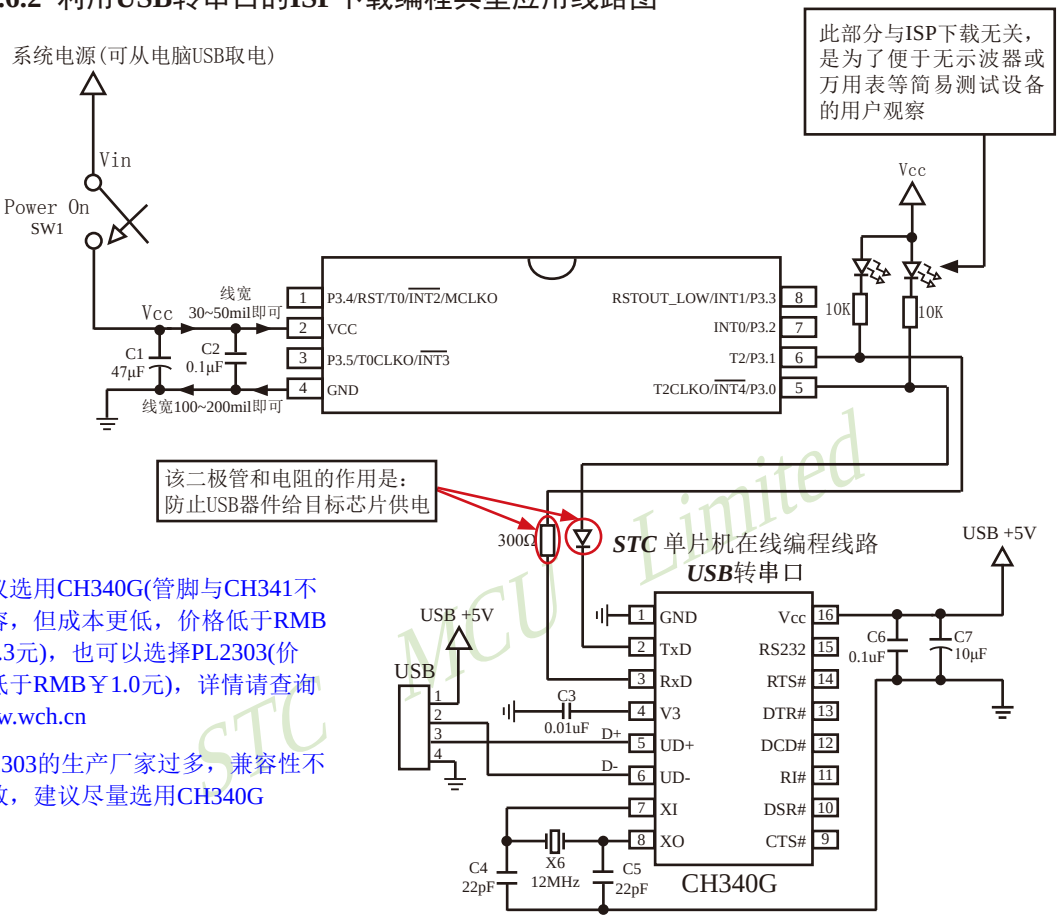
内部高可靠复位，可彻底省掉外部复位电路

P3.4/RST/T0/$\overline{\text{INT2}}$/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.3.6.2 利用USB转串口的ISP下载编程典型应用线路图



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

内部高可靠复位，可彻底省掉外部复位电路

P3.4/RST/T0/INT2/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.3.7 STC15W104系列单片机的管脚说明

管脚	管脚编号 (封装SOP-8/DIP-8)	说明	
P3.0/$\overline{\text{INT4}}$ /T2CLKO	5	P3.0	标准I/O口 PORT3[0]
		$\overline{\text{INT4}}$	外部中断4, 只能下降沿中断 $\overline{\text{INT4}}$支持掉电唤醒
		T2CLKO	T2的时钟输出 可通过设置INT_CLKO[2]位/T2CLKO将该管脚配置为T2CLKO
P3.1/T2	6	P3.1	标准I/O口 PORT3[1]
		T2	定时器/计数器2的外部输入
P3.2/INT0	7	P3.2	标准I/O口 PORT3[2]
		INT0	外部中断0，既可上升沿中断也可下降沿中断. 如果IT0(TCON.0)被置为1, INT0管脚仅为下降沿中断。如果IT0(TCON.0)被清0，INT0管脚既支持上升沿中断也支持下降沿中断。 INT0支持掉电唤醒。
P3.3/INT1/ RSTOUT_LOW	8	P3.3	标准I/O口 PORT3[3]
		INT1	外部中断1，既可上升沿中断也可下降沿中断. 如果IT1(TCON.2)被置为1, INT1管脚仅为下降沿中断。如果IT1(TCON.2)被清0，INT1管脚既支持上升沿中断也支持下降沿中断。 INT1支持掉电唤醒。
		RSTOUT_LOW	上电后, 输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P3.4/RST/T0/$\overline{\text{INT2}}$ /MCLKO	1	P3.4	标准I/O口 PORT3[4]
		RST	复位脚, 高电平复位
		T0	定时器/计数器0的外部输入
		$\overline{\text{INT2}}$	外部中断2, 只能下降沿中断 $\overline{\text{INT2}}$支持掉电唤醒
		MCLKO	主时钟输出;输出的频率可为MCLK/1, MCLK/2, MCLK/4（MCLK为主时钟频率）。 此系列的主时钟只可以是内部R/C时钟。
P3.5/T0CLKO/ $\overline{\text{INT3}}$	3	P3.5	标准I/O口 PORT3[5]
		T0CLKO	定时器/计数器0的时钟输出 可通过设置INT_CLKO[0]位/T0CLKO将该管脚配置为T0CLKO, 也可对T0脚的外部时钟输入进行分频输出
		$\overline{\text{INT3}}$	外部中断3, 只能下降沿中断 $\overline{\text{INT3}}$支持掉电唤醒
Vcc	2	电源正极	
Gnd	4	电源负极，接地	

STC MCU Limited

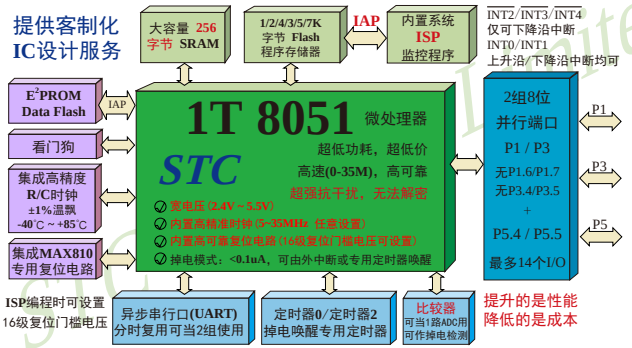
1.4 STC15W201S系列单片机总体介绍(A版大批量现货供应中)

1.4.1 STC15W201S系列单片机简介

STC15W201S系列单片机是STC生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/超强抗干扰的新一代8051单片机，采用STC第九代加密技术，无法解密，指令代码完全兼容传统8051,但速度快8-12倍。内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP编程时16级复位门槛电压可选)。1组高速异步串行通信口(UART)，可在2组管脚之间进行切换，分时复用可作2组串口使用，针对串行口通信/电机控制/强干扰场合。内置比较器，功能更强大。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通8051快8-12倍
2. 工作电压：2.4V - 5.5V
3. 1K/2K/3K/4K/5K/7.5K字节片内Flash程序存储器，擦写次数10万次以上
4. 片内集成256字节的SRAM
5. 有片内EEPROM功能，擦写次数10万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 内部高可靠复位，ISP编程时16级复位门槛电压可选，可彻底省掉外部复位电路
8. 内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时内部时钟从5MHz~35MHz可设(5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz)
9. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
10. 工作频率范围：5MHz ~ 35MHz，相当于普通8051的60MHz～420MHz
11. 一组高速异步串行通信端口，可在2组管脚之间进行切换，分时复用可当2组串口使用：串行口(RxD/P3.0, TxD/P3.1)可以切换到(RxD_2/P3.6, TxD_2/P3.7).
12. 支持程序加密后传输，防拦截
13. 支持RS485下载

14.低功耗设计：低速模式，空闲模式，掉电模式/停机模式。

15.可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器

16.可将掉电模式/停机模式唤醒的资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚RxD(可在RxD/P3.0和RxD_2/P3.6之间切换)；管脚T0/T2(下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；内部低功耗掉电唤醒专用定时器。

17. 共2个定时器/计数器，分别是16位可重装载的定时器/计数器0(即T0)和定时器/计数器2(即T2)，并都可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4）。

18. 可编程时钟输出功能(对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出)：
由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz.；
而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz

① T0在P3.5/T0CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；

② T2在P3.0/T2CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；

以上2个定时器/计数器均可1~65536级分频输出。

③ 主时钟在P5.4/MCLKO对外输出时钟，并可如下分频MCLK/1, MCLK/2, MCLK/4.

STC15W201S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。MCLK是指主时钟频率，MCLKO是指主时钟输出。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/P1.6口对外输出主时钟。

19. 比较器，可当1路ADC使用，并可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出（可设置极性），也支持外部管脚CMP+与内部参考电压进行比较

20. 硬件看门狗(WDT)

21. 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令

22. 通用I/O口（14/6个），复位后为：准双向口/弱上拉(普通8051传统I/O口)，
可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
每个I/O口驱动能力均可达到20mA，但整个芯片电流最大不要超过90mA.
如果I/O口不够用，可外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口。

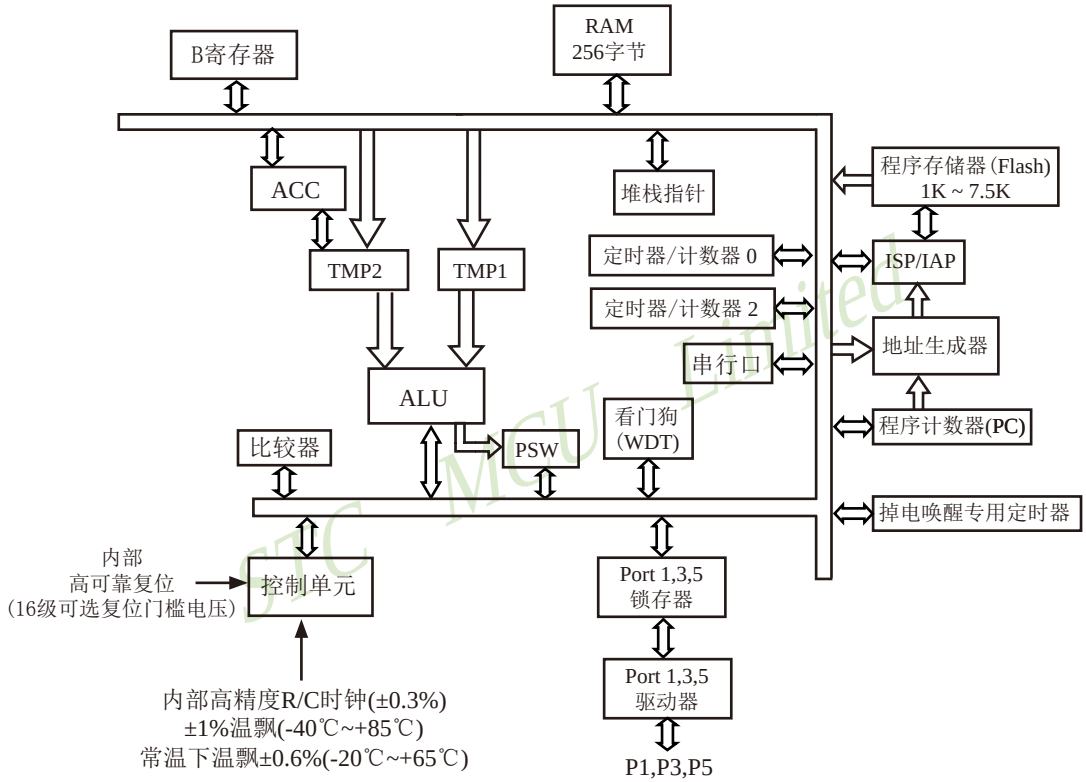
23. 封装：SOP8, SOP16 (6mm x 9mm), DIP16.

24. 全部175℃八小时高温烘烤，高品质制造保证

25. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

1.4.2 STC15W201S系列单片机的内部结构图

STC15W201S系列单片机的内部结构框图如下图所示。STC15W201S系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、掉电唤醒专用定时器、I/O口、1组高速异步串行通信端口、比较器、看门狗、片内高精度R/C时钟及高可靠复位等模块。



STC15W201S系列内部结构框图

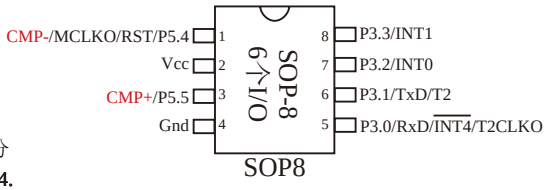
1.4.3 STC15W201S系列单片机管脚图

所有封装形式均满足欧盟RoHS要求，强烈推荐选择SOP16贴片封装，传统的插件DIP16封装稳定供货。

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

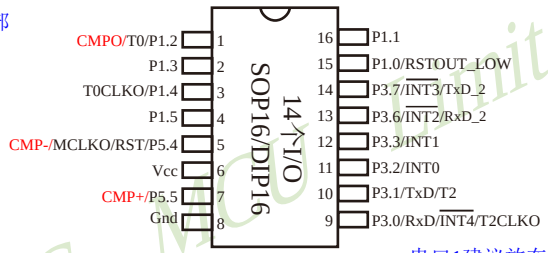
对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；



SOP8

MCLKO是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4.

此系列的主时钟只可以是内部R/C时钟。MCLK是指主时钟频率。



SOP16/DIP16

串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]上。

T0CLKO是指定时器/计数器0的可编程时钟输出
(对内部系统时钟或对外部管脚T0/P1.2的时钟输入进行可编程时钟分频输出);

T2CLKO是指定时器/计数器2的可编程时钟输出
(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出);

T0CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T2的时钟输入进行时钟分频输出，作分频器使用。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	x0xx,xx0x
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	00x0,x000

串口1/S1可在2个地方切换，由 S1_S0 控制位来选择	
S1_S0	串口1/S1可在P1/P3之间来回切换
0	串口1/S1在[P3.0/RxD,P3.1/TxD]
1	串口1/S1在[P3.6/RxD_2,P3.7/TxD_2]

串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]上。

建议用户在程序中将[S1_S0]的值设置为1，进而将串口1放在[P3.6/RxD_2, P3.7/TxD_2]上

STC MCU Limited

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	00x0,x000

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， 输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， 输出时钟频率 = MCLK / 4

STC15W201S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟，MCLK是指主时钟频率。
STC15W201S系列单片机在MCLKO/P5.4口对外输出主时钟。
STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列)均在MCLKO/P5.4口对外输出主时钟。

Tx_Rx：串口1的中继广播方式设置
　0：串口1为正常工作方式
　1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

串口1的RxD管脚和TxD管脚可以在2组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1];
[RxD_2/P3.6, TxD_2/P3.7].

STC MCU Limited

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给CPU、串行口、定时器的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

STC15W201S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。

1.4.4 STC15W201S系列单片机选型价格一览表

型号	工作电压（V）	Flash程序存储器(byte)	SRAM字节	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器(可当1路A/D使用,可作外部掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	所有封装 SOP8 SOP16/DIP16 价格(RMB ¥)		
																					SOP8	SOP16	DIP16
STC15W201S系列单片机选型价格一览表																							
STC15W201S	5.5-2.4	1K	256	1	–	2	-	有	5	–	有	1	4K	有	有	16级	有	是	有	是	¥1.3	¥1.4	¥1.6
STC15W202S	5.5-2.4	2K	256	1	–	2	-	有	5	–	有	1	3K	有	有	16级	有	是	有	是	¥1.4	¥1.5	¥1.7
STC15W203S	5.5-2.4	3K	256	1	–	2	-	有	5	–	有	1	2K	有	有	16级	有	是	有	是	¥1.5	¥1.6	¥1.8
STC15W204S	5.5-2.4	4K	256	1	–	2	-	有	5	–	有	1	1K	有	有	16级	有	是	有	是	¥1.6	¥1.7	¥1.9
IAP15W205S	5.5-2.4	5K	256	1	–	2	-	有	5	–	有	1	IAP	有	有	16级	有	是	有	是	¥1.6	¥1.7	¥1.9
																					用户可将用户程序区的程序FLASH当EEPROM使用		
IRC15W207S 固定使用内部24MHz时钟	5.5-2.4	7.5K	256	1	–	2	-	有	5	–	有	1	IAP	有	有	固定	有	是	无	否	¥1.6	¥1.7	¥1.9
																					用户可将用户程序区的程序FLASH当EEPROM使用		

提供客制化IC服务

如果要用16-pin单片机，建议用户选用SOP16封装。

以上单价为10K起订
量小每片需加0.1元
以上价格运费由客户承担,零售10片起
如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

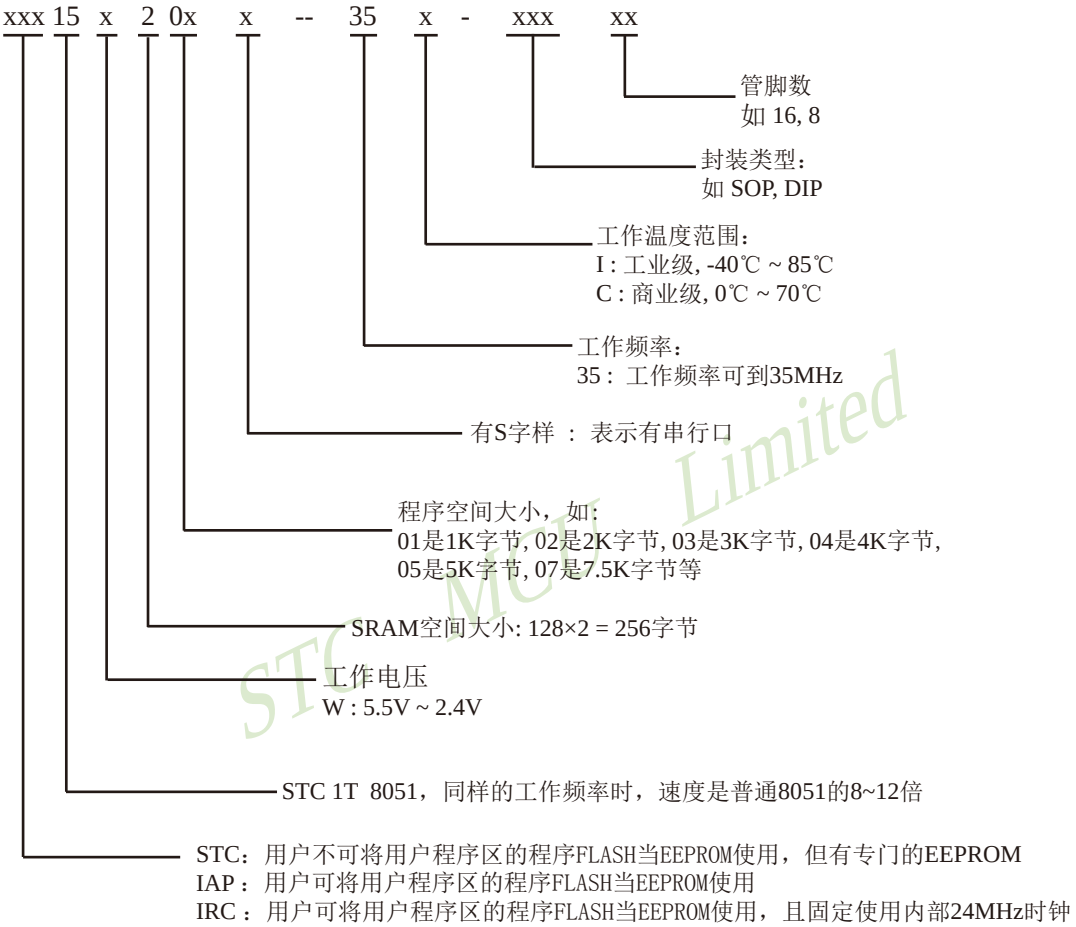
上表中IRC15W207S型号的单片机固定使用内部24MHz时钟，且其内部复位门槛电压固定，同时不支持"程序加密后传输"功能，其P5.4不可当复位管脚RST使用，且P3.2/P3.3与下载无关。

总结：STC15W201S系列单片机有：2普通定时器/计数器(这2个普通定时器/计数器是指：T0和T2)；　掉电唤醒专用定时器；5个支持掉电唤醒的外部中断INT0/INT1/INT2/INT3/INT4；1组高速异步串行通信端口；1个比较器；1个数据指针DPTR等功能。表中"-"表示该型号的单片机无相应的功能。STC15W201S系列单片机无SPI、无A/D转换、无CCP/PWM/PCA、无外部数据总线等功能。

因为程序区的最后7个字节单元被强制性的放入全球唯一ID号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少7个字节。

STC MCU Limited

1.4.5 STC15W201S系列单片机命名规则



命名举例：

（1）STC15W201S- 35I - SOP16 表示：
用户不可将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为256字节，程序空间大小为1K，有一组串行口，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为16。

（2）STC15W201S- 35I - DIP16 表示：
用户不可将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为256字节，程序空间大小为1K，有一组串行口，工作频率

可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为DIP封装，管脚数为16。

（3）IAP15W205S- 35I - SOP16 表示：

用户可将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为256字节，程序空间大小为5K，有一组串行口，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为16。

（4）IAP15W205S- 35I - DIP16 表示：

用户可将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为256字节，程序空间大小为5K，有一组串行口，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为DIP封装，管脚数为16。

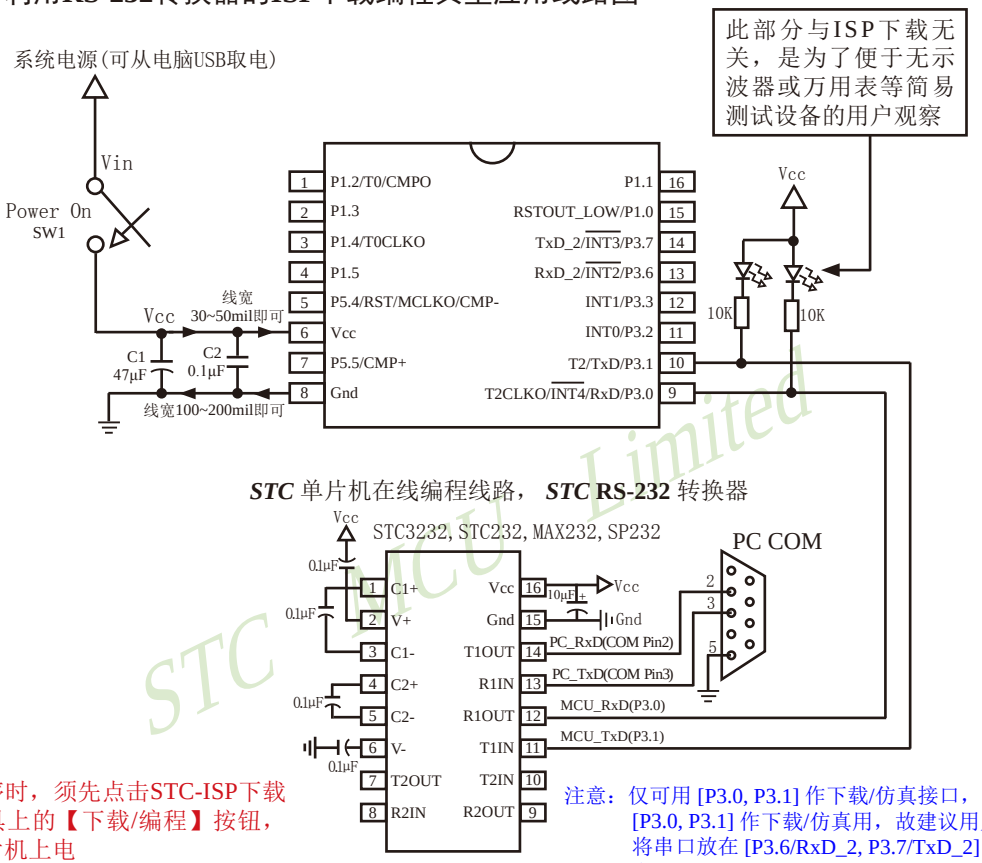
（5）IRC15W207S- 35I - DIP16 表示：

用户可将用户程序区的程序FLASH当EEPROM使用，且固定使用内部24MHz时钟，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为256字节，程序空间大小为7.5K，有一组串行口，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为DIP封装，管脚数为16。

STC MCU Limited

1.4.6 STC15W201S系列单片机在系统可编程(ISP)典型应用线路图

1.4.6.1 利用RS-232转换器的ISP下载编程典型应用线路图



烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 上。

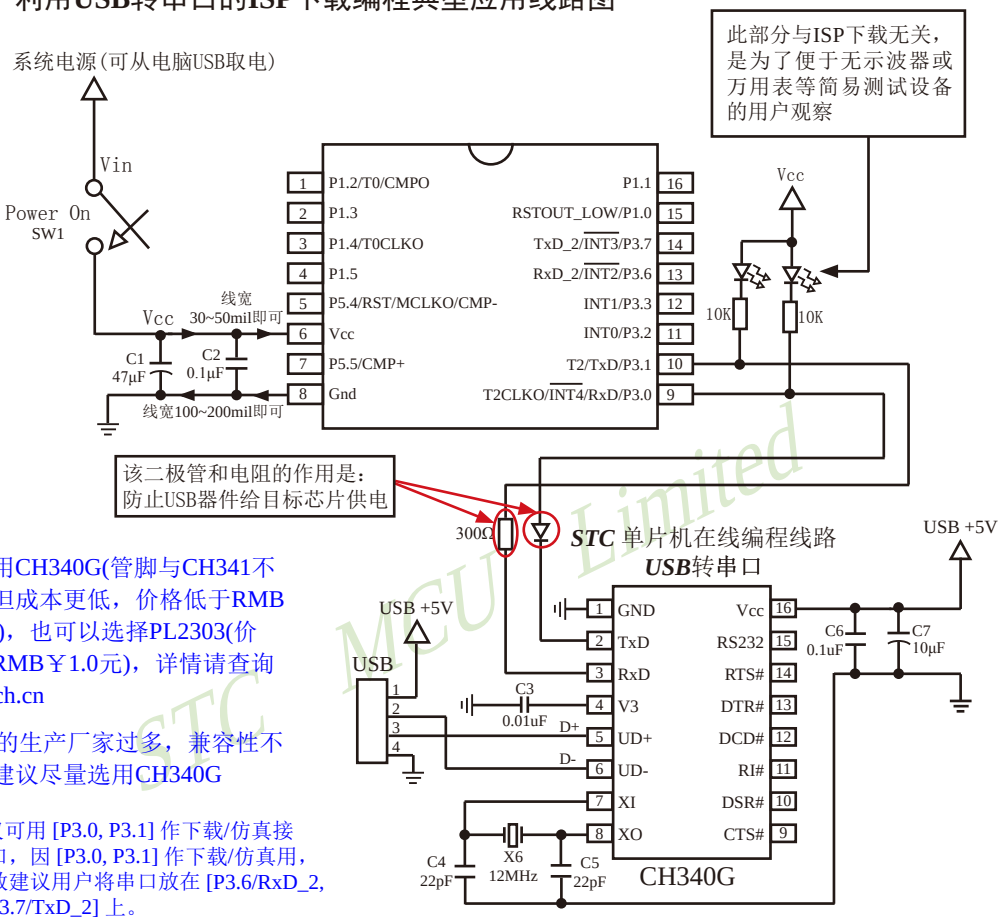
内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.4.6.2 利用USB转串口的ISP下载编程典型应用线路图



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 上。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.4.7 STC15W201S系列单片机的管脚说明

管脚	封装		说明	
	SOP8	SOP16/DIP16		
P1.0/RSTOUT_LOW		15	P1.0	标准I/O口 PORT1[0]
			RSTOUT_LOW	上电后,输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P1.1		16	标准I/O口 PORT1[1]	
P1.2/T0/CMPO		1	P1.2	标准I/O口 PORT1[2]
			T0	定时器/计数器1的外部输入
			CMPO	比较器的比较结果输出管脚
P1.3		2	标准I/O口 PORT1[3]	
P1.4/T0CLKO		3	P1.4	标准I/O口 PORT1[4]
			T0CLKO	定时器/计数器0的时钟输出 可通过设置INT_CLKO[0]位/T0CLKO将该管脚配置为T0CLKO，也可对T0脚的外部时钟输入进行分频输出
P1.5		4	标准I/O口 PORT1[5]	
P3.0/RxD/$\overline{\text{INT4}}$/T2CLKO	5	9	P3.0	标准I/O口 PORT3[0]
			RxD	串口数据接收端
			$\overline{\text{INT4}}$	外部中断4,只能下降沿中断, $\overline{\text{INT4}}$支持掉电唤醒
			T2CLKO	T2的时钟输出 可通过设置INT_CLKO[2]位/T2CLKO将该管脚配置为T2CLKO
P3.1/TxD/T2	6	10	P3.1	标准I/O口 PORT3[1]
			TxD	串口数据发送端
			T2	定时器/计数器2的外部输入
P3.2/INT0	7	11	P3.2	标准I/O口 PORT3[2]
			INT0	外部中断0，既可上升沿中断也可下降沿中断. 如果IT0(TCON.0)被置为1,INT0管脚仅为下降沿中断。如果IT0(TCON.0)被清0，INT0管脚既支持上升沿中断也支持下降沿中断。 INT0支持掉电唤醒。
P3.3/INT1	8	12	P3.3	标准I/O口 PORT3[3]
			INT1	外部中断1，既可上升沿中断也可下降沿中断. 如果IT1(TCON.2)被置为1,INT1管脚仅为下降沿中断。如果IT1(TCON.2)被清0，INT1管脚既支持上升沿中断也支持下降沿中断。 INT1支持掉电唤醒。
P3.6/$\overline{\text{INT2}}$/RxD_2		13	P3.6	标准I/O口 PORT3[6]
			$\overline{\text{INT2}}$	外部中断2,只能下降沿中断 $\overline{\text{INT2}}$支持掉电唤醒
			RxD_2	串口数据接收端
P3.7/$\overline{\text{INT3}}$/TxD_2		14	P3.7	标准I/O口 PORT3[7]
			$\overline{\text{INT3}}$	外部中断3,只能下降沿中断 $\overline{\text{INT3}}$支持掉电唤醒
			TxD_2	串口数据发送端

STC Micro Limited

管脚	封装		说明	
	SOP8	SOP16/ DIP16		
P5.4/RST/ MCLKO/CMP-	1	5	P5.4	标准I/O口 PORT5[4]
			RST	复位脚(高电平复位)
			MCLKO	主时钟输出;输出的频率可为MCLK/1,MCLK/2, MCLK/4（MCLK是指主时钟频率）。 此系列的主时钟只可以是内部R/C时钟，MCLK指主时钟频率。
			CMP-	比较器负极输入端
P5.5/CMP+	3	7	P5.5	标准I/O口 PORT5[5]
			CMP+	比较器正极输入端
Vcc	2	6	电源正极	
Gnd	4	8	电源负极，接地	

STC MCU Limited

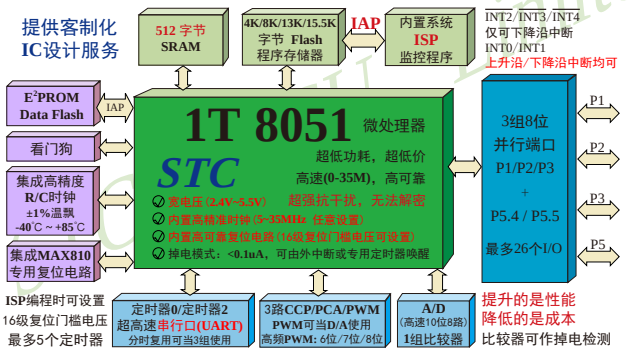
1.5 STC15W408AS系列单片机总体介绍(大批量现货供应中)

1.5.1 STC15W408AS系列单片机简介

STC15W408AS系列单片机是STC生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/超强抗干扰的新一代8051单片机，采用STC第九代加密技术，无法解密，指令代码完全兼容传统8051,但速度快8-12倍。内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP编程时16级复位门槛电压可选)。3路CCP/PWM/PCA，8路高速10位A/D转换(30万次/秒)，1组超高速异步串行通信口(UART，可在3组管脚之间进行切换，分时复用可作3组串口使用)，1组高速同步串行通信端口SPI，针对串行口通信/电机控制/强干扰场合。内置比较器，功能更强大。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通8051快8-12倍
2. 工作电压：5.5V - 2.4V
3. 4K / 8K / 10K / 12K / 13K / 15.5K字节片内Flash程序存储器，擦写次数10万次以上
4. 片内集成512字节的SRAM，包括常规的256字节RAM <idata> 和内部扩展的256字节XRAM <xdata>
5. 有片内EEPROM功能，擦写次数10万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 共8通道10位高速ADC，速度可达30万次/秒，3路PWM还可当3路D/A使用
8. 共3通道捕获/比较单元(CCP/PWM/PCA)
   ----也可用来再实现3个定时器或3个外部中断(支持上升沿/下降沿中断)或3路D/A
9. 利用CCP/PCA高速脉冲输出功能可实现3路9~16位PWM (每通道占用系统时间小于0.6%)
10. 利用定时器T0的时钟输出功能可实现高精度的8~16位PWM (占用系统时间小于0.4%)
11. 内部高可靠复位，ISP编程时16级复位门槛电压可选，可彻底省掉外部复位电路

10.内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时内部时钟从5MHz~35MHz可设(5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz)

11.不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号

12.工作频率范围：5MHz ~ 35MHz，相当于普通8051的60MHz～420MHz

13.一组高速异步串行通信端口，可在3组管脚之间进行切换，分时复用可当3组串口使用：
串行口(RxD/P3.0, TxD/P3.1)可以切换到(RxD_2/P3.6, TxD_2/P3.7),
还可以切换到(RxD_3/P1.6, TxD_3/P1.7)；

14.一组高速同步串行通信端口SPI。

15.支持程序加密后传输，防拦截

16.支持RS485下载

17.低功耗设计：低速模式，空闲模式，掉电模式/停机模式。

18. 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。

19.可将掉电模式/停机模式唤醒的资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚CCP0/CCP1/CCP2；管脚RxD(可在RxD/P3.0、RxD_2/P3.6和RxD_3/P1.6之间切换)；管脚T0/T2(下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；内部低功耗掉电唤醒专用定时器。

STC MCU Limited

20. 共5个定时器/计数器，2个16位可重装载定时器/计数器(T0/T2，其中T0兼容普通8051的定时器/计数器)，并均可独立实现对外可编程时钟输出(2通道)，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4），3路CCP/PWM/PCA还可再实现3个定时器

21. 可编程时钟输出功能(对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出)：
由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz.；
而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz.
① T0在P3.5/T0CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；
② T2在P3.0/T2CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；
以上2个定时器/计数器均可1~65536级分频输出。

③ 主时钟在P5.4/MCLKO或P1.6/XTAL2/MCLKO_2对外输出时钟，并可如下分频MCLK/1, MCLK/2, MCLK/4.

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。MCLK是指主时钟频率，MCLKO是指主时钟输出。

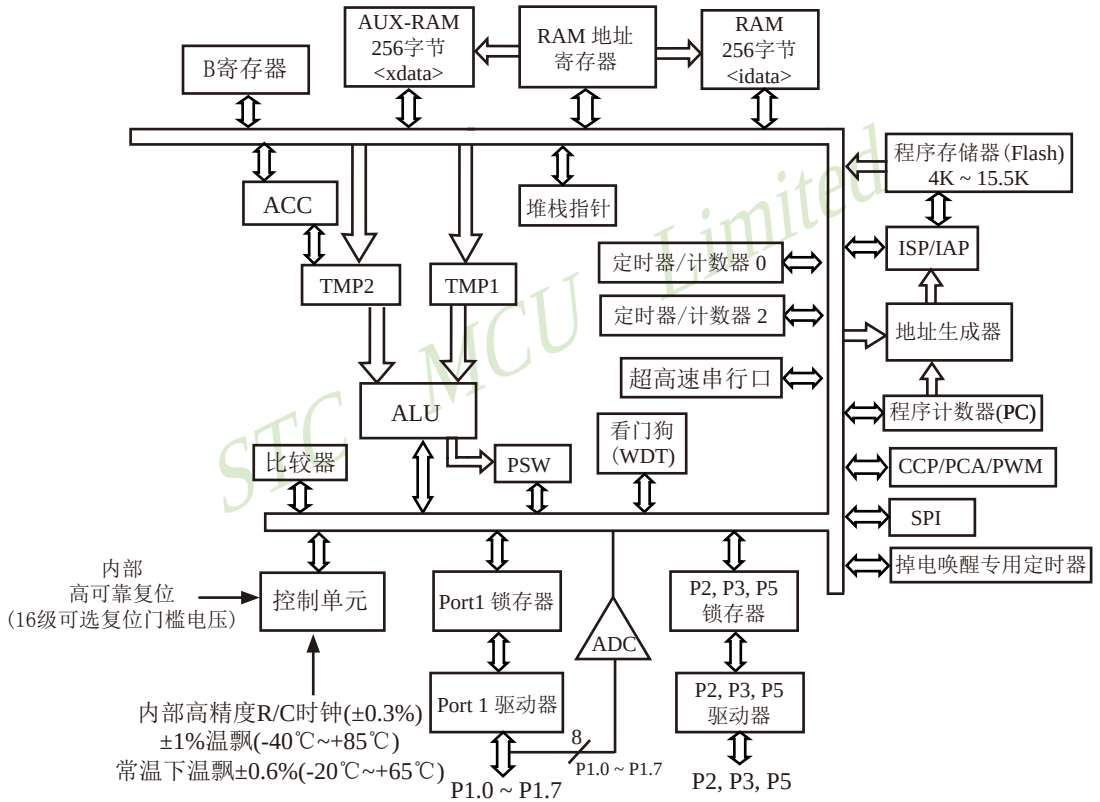
STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W408AS系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

22. 比较器，可当1路ADC使用，并可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出（可设置极性），也支持外部管脚CMP+与内部参考电压进行比较
23. 硬件看门狗(WDT)
24. 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
25. 通用I/O口（26/18/14个），复位后为：准双向口/弱上拉（普通8051传统I/O口）
    可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
    每个I/O口驱动能力均可达到20mA，但整个芯片电流最大不要超过90mA.
    如果I/O口不够用，可外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口
26. 封装：SOP28, TSSOP28(6.4mm x 9.7mm), QFN28(5mm x 5mm), SKDIP28, SOP20, TSSOP20(6.5mm x 6.5mm), DIP20, SOP16, DIP16.
27. 全部175℃八小时高温烘烤，高品质制造保证
28. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

STC MCU Limited

1.5.2 STC15W408AS系列单片机的内部结构图

STC15W408AS系列单片机的内部结构框图如下图所示。STC15W408AS系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、掉电唤醒专用定时器、I/O口、高速A/D转换(30万次/秒)、比较器、看门狗、高速异步串行通信端口UART、CCP/PWM/PCA、高速同步串行端口SPI，片内高精度R/C时钟及高可靠复位等模块。STC-15W408AS系列单片机几乎包含了数据采集和控制中所需的所有单元模块，可称得上是一个片上系统(SysTem Chip或SysTem on Chip,简写为STC,这是宏晶科技STC名称的由来)。



STC15W408AS系列内部结构框图

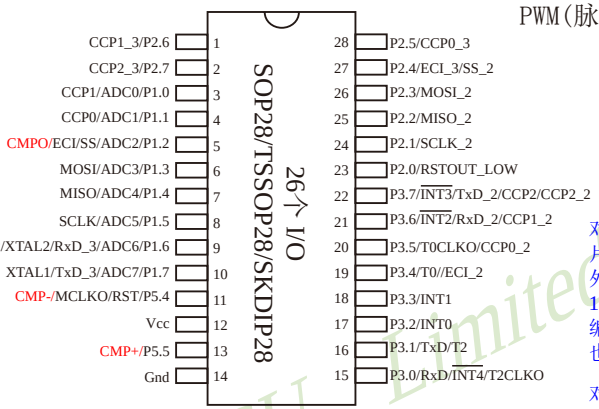
1.5.3 STC15W408AS系列单片机管脚图

所有封装形式均满足欧盟RoHS要求，强烈推荐选择SOP28/SOP20/SOP16贴片封装，传统的插件SKDIP28/DIP20/DIP16封装稳定供货。

CCP：是英文单词的缩写
Capture(捕获),
Compare(比较),
PWM(脉宽调制)

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

P1.x/ADCx是指P1.x管脚可作为A/D转换通道使用



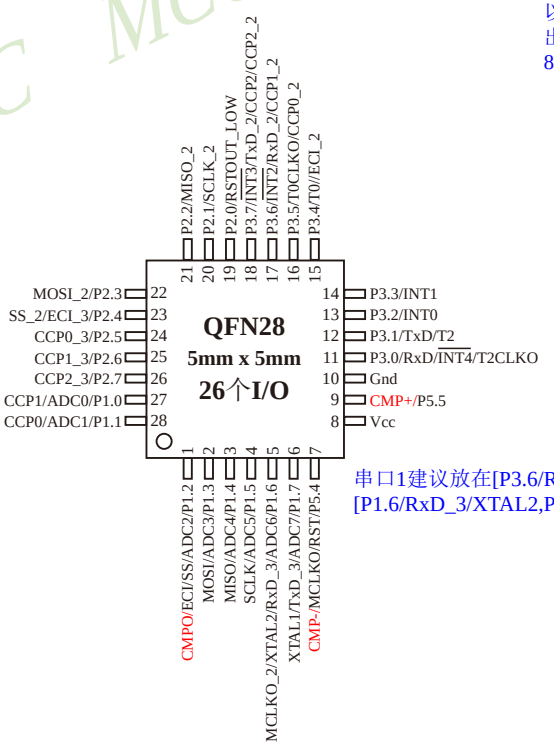
SOP28/SKDIP28

MCLKO_2与MCLKO均是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4.

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。MCLK是指主时钟频率。

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；

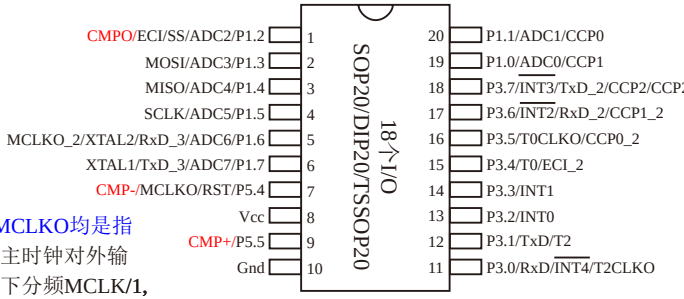


T0CLKO是指定时器/计数器0的可编程时钟输出
(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；
T2CLKO是指定时器/计数器2的可编程时钟输出
(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；

T0CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T2的时钟输入进行时钟分频输出，作分频器使用。

串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1]上。

STC MCU Limited

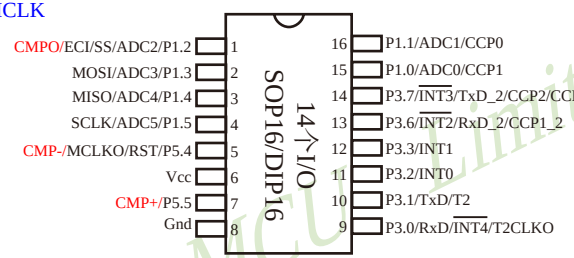


SOP20/DIP20/TSSOP20

MCLKO_2与MCLKO均是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4.

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。MCLK是指主时钟频率。

串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1]上。



SOP16/DIP16

STC MCU Limited

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,x00x
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

串口1/**S1**可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD, P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2, P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1] 串口1在P1口时要使用内部时钟
1	1	无效

串口1建议放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1]上。

建议用户在程序中将[S1_S1, S1_S0]的值设置为[0, 1]或[1, 0]，进而将串口1放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1]上

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,x00x
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

CCP可在3个地方切换，由 CCP_S1 及 CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP可在P1/P2/P3之间来回切换
0	0	CCP在[P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2]
0	1	CCP在[P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2]
1	0	CCP在[P2.4/ECI_3,P2.5/CCP0_3,P2.6/CCP1_3,P2.7/CCP2_3]
1	1	无效

SPI可在2个地方切换，由 SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI可在P1/P2之间来回切换
0	0	SPI在[P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK]
0	1	SPI在[P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2]
1	0	SPI在[P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	无效

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， 输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， 输出时钟频率 = MCLK / 4

主时钟可以是内部R/C时钟,也可是外部输入的时钟或外部晶体振荡产生的时钟,MCLK是指主时钟频率。
STC15W408AS系列单片机在MCLKO/P5.4口或MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列)均在MCLKO/P5.4口对外输出主时钟，且STC15W408AS系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15W408AS系列单片机通过CLK_DIV.3/MCLKO_2位来选择是在MCLKO/P5.4口对外输出主时钟，还是在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位
　　0：在MCLKO/P5.4口对外输出主时钟；
　　1：在MCLKO_2/XTAL2/P1.6口对外输出主时钟；
主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

STC MCU Limited

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D转换的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

ADRJ：ADC转换结果调整

0：ADC_RES[7:0]存放高8位ADC结果，ADC_RESL[1:0]存放低2位ADC结果

1：ADC_RES[1:0]存放高2位ADC结果，ADC_RESL[7:0]存放低8位ADC结果

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

串口1的RxD管脚和TxD管脚可以在3组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1];
[RxD_2/P3.6, TxD_2/P3.7];
[RxD_3/P1.6, TxD_3/P1.7].

STC MCU Limited

1.5.4 STC15W408AS系列单片机选型价格一览表

型号	工作电压（V）	Flash程序存储器(byte)	SRAM字节	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器(可当1路A/D使用,可作外部掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	所有封装 SOP28/TSSOP28/ SKDIP28/QFN28 SOP20 / DIP20 / TSSOP20 SOP16 / DIP16		
																					部分封装价格(RMB ¥)		
																					SOP28	SOP20	SOP16
STC15W408AS系列单片机选型价格一览表 特别提示：3路CCP/PCA/PWM还可当3路定时器使用																							
STC15W401AS	2.4-5.5	1K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16级	有	是	有	是	¥2.00	¥1.80	¥1.60
STC15W402AS	2.4-5.5	2K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16级	有	是	有	是	¥2.20	¥2.00	¥1.80
STC15W404AS	5.5-2.4	4K	512	1	有	2	3-ch	有	5	10-bit	有	1	9K	有	有	16级	有	是	有	是	¥2.50	¥2.20	¥1.99
STC15W408AS	5.5-2.4	8K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16级	有	是	有	是	¥2.70	¥2.40	¥2.20
IAP15W413AS	5.5-2.4	13K	512	1	有	2	3-ch	有	5	10-bit	有	1	IAP	有	有	16级	有	是	有	是	¥2.80	¥2.50	¥2.30
																					用户可将用户程序区的程序FLASH当EEPROM使用		
IRC15W415AS 使用外部晶振或内部24MHz时钟	5.5-2.4	15.5K	512	1	有	2	3-ch	有	5	10-bit	有	1	IAP	有	有	固定	有	是	无	否	¥2.80	¥2.50	¥2.30
																					用户可将用户程序区的程序FLASH当EEPROM使用		

以上单价为10K起订，量小每片需加0.1元，以上价格运费由客户承担,零售10片起，如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

IRC15W415AS型号单片机的内部复位门槛电压固定，P5.4不可当复位管脚RST使用，[XTAL2/P1.6, XTAL1/P1.7]不可当I/O口使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能 。

如果要用28-pin单片机，建议用户选用SOP28封装；
如果要用16-pin单片机，建议用户选用SOP16封装。
如果要用20-pin单片机，建议用户选用SOP20封装。
提供客制化IC服务

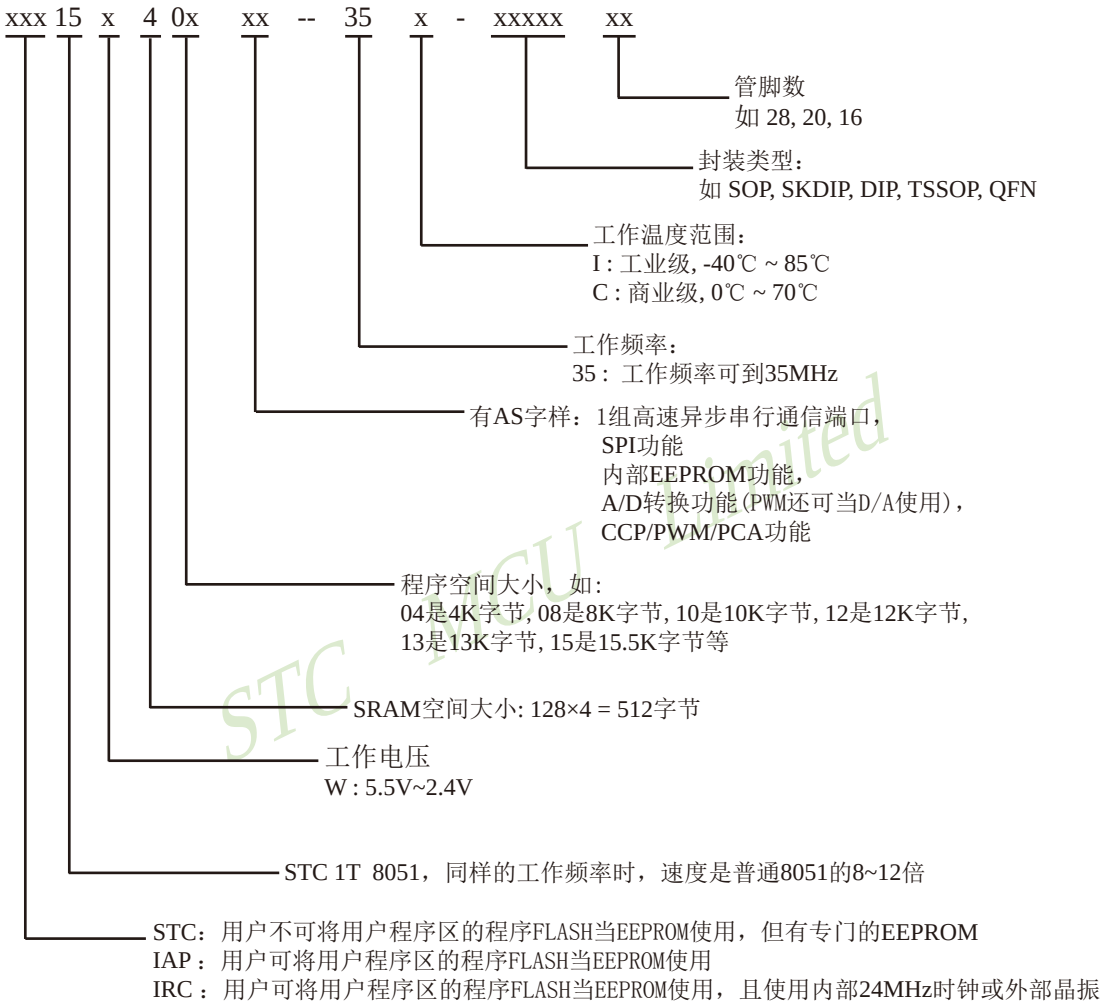
因为程序区的最后7个字节单元被强制性的放入全球唯一ID号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少7个字节。

总结：STC15W408AS系列单片机（含IRC15W415AS）有：2普通定时器/计数器(T0和T2)，3路CCP/PWM/PCA(可再实现3个定时器使用)；掉电唤醒专用定时器；5个支持掉电唤醒的外部中断INT0/INT1/INT2/INT3/INT4；1组高速异步串行通信端口；1组高速同步串行通信端口SPI；8路高速10位A/D转换器；1个比较器；1个数据指针DPTR等功能。STC15W408AS系列单片机没有外部数据总线。

1.4.5 STC15W408AS系列单片机封装价格一览表

型号	工作电压（V）	工作频率(MHz)	工作温度(I — 工业级)	所有封装 SOP28/ TSSOP28/ SKDIP28/ QFN28/ SOP20/ DIP20/ TSSOP20/ SOP16/ DIP16								
				SOP28	TSSOP28	SKDIP28	QFN28	SOP20	TSSOP20	DIP20	SOP16	DIP16
STC15W408AS系列单片机封装价格一览表												
STC15W401AS	5.5-2.4	35	-40℃ ~ +85℃	¥2.00	¥2.10	¥2.20	¥2.20	¥1.80	¥1.90	¥2.00	¥1.60	¥1.80
STC15W402AS	5.5-2.4	35	-40℃ ~ +85℃	¥2.20	¥2.30	¥2.40	¥2.40	¥2.00	¥2.10	¥2.20	¥1.80	¥2.00
STC15W404AS	5.5-2.4	35	-40℃ ~ +85℃	¥2.50	¥2.60	¥2.70	¥2.70	¥2.20	¥2.30	¥2.40	¥1.99	¥2.20
STC15W408AS	5.5-2.4	35	-40℃ ~ +85℃	¥2.70	¥2.80	¥2.90	¥2.90	¥2.40	¥2.50	¥2.60	¥2.20	¥2.40
IAP15W413AS	5.5-2.4	35	-40℃ ~ +85℃	¥2.80	¥2.90	¥3.00	¥3.00	¥2.50	¥2.60	¥2.70	¥2.30	¥2.50
IRC15W415AS	5.5-2.4	35	-40℃ ~ +85℃	¥2.80	¥2.90	¥3.00	¥3.00	¥2.50	¥2.60	¥2.70	¥2.30	¥2.50

1.5.6 STC15W408AS系列单片机命名规则



命名举例：

（1）STC15W404AS- 35I - SOP16 表示：

用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为512字节，程序空间大小为4K，有1组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为16。

（2）STC15W408AS- 35I - SOP20 表示：

用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为512字节，程序空间大小为8K，有1组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为20。

（3）STC15W412AS- 35I - SOP28 表示：

用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为512字节，程序空间大小为12K，有1组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为28。

（4）IAP15W413AS- 35I - SOP28 表示：

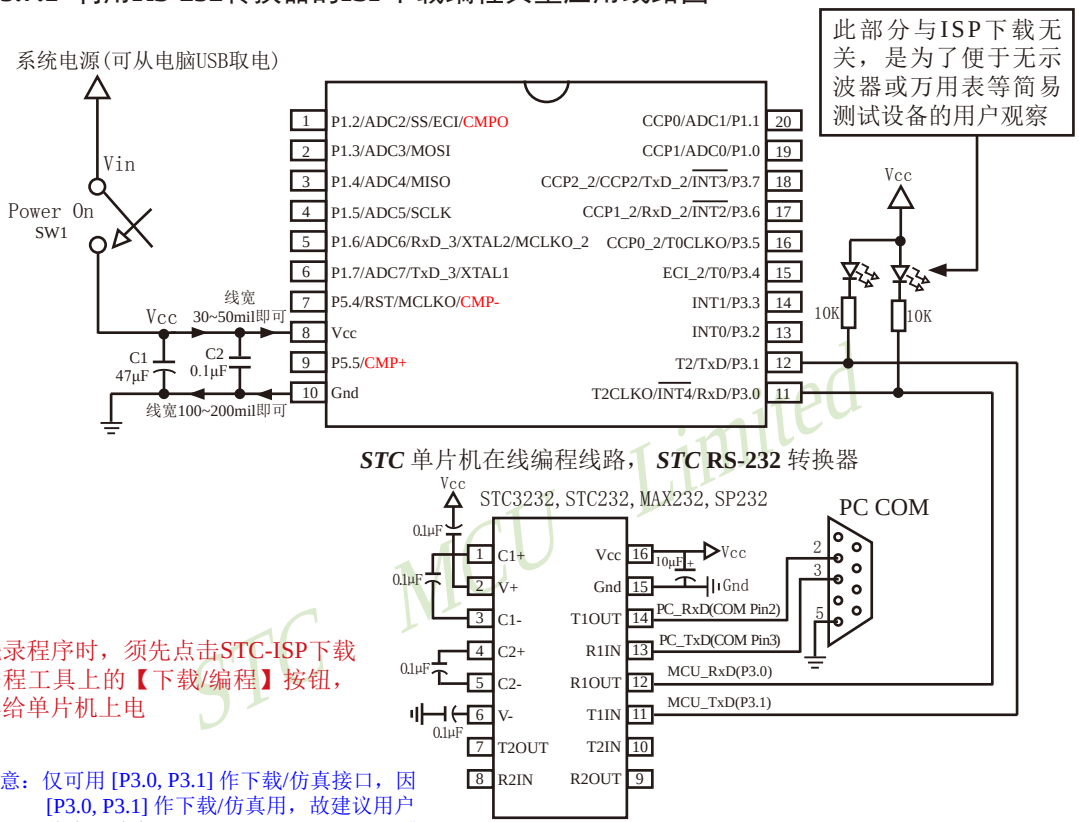
用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为512字节，程序空间大小为13K，有1组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为28。

（5）IAP15W413AS- 35I - SKDIP28 表示：

用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为512字节，程序空间大小为13K，有1组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SKDIP封装，管脚数为28。

1.5.7 STC15W408AS系列单片机在系统可编程(ISP)典型应用线路图

1.5.7.1 利用RS-232转换器的ISP下载编程典型应用线路图



烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

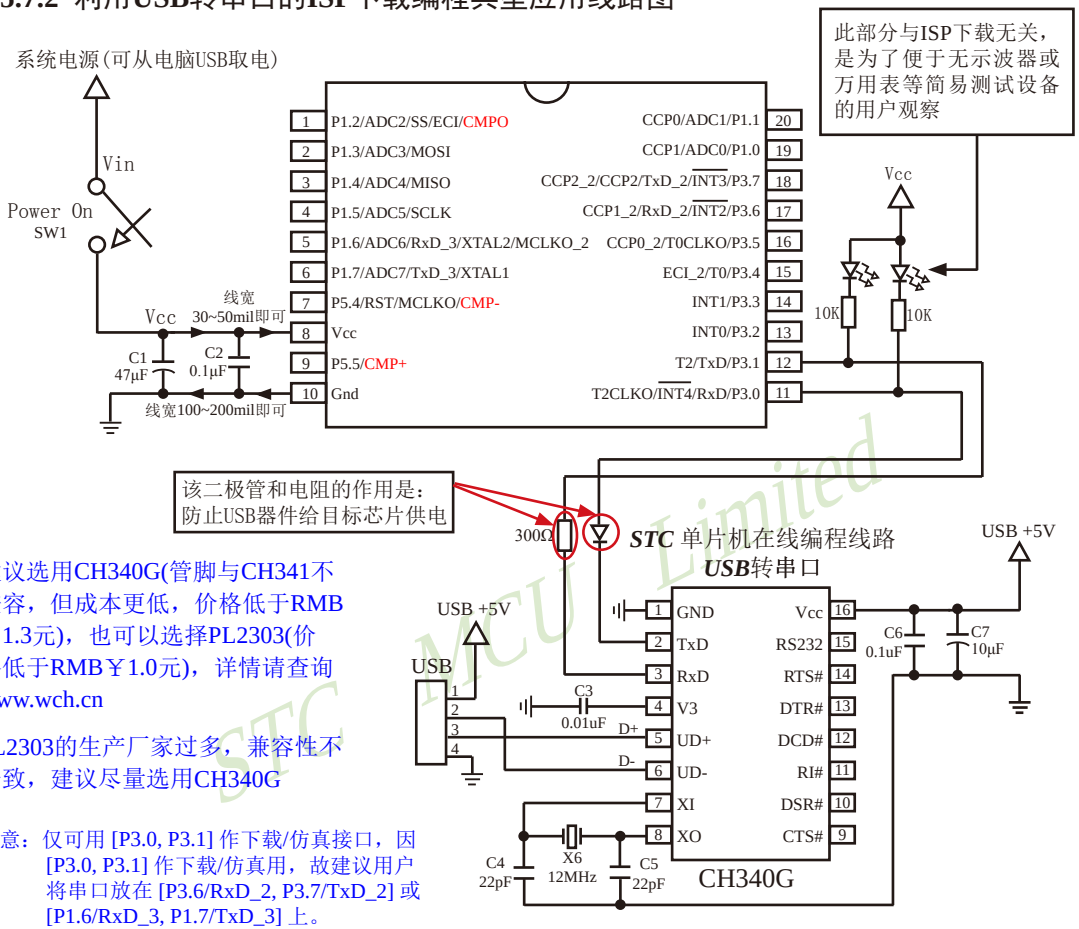
内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.5.7.2 利用USB转串口的ISP下载编程典型应用线路图



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.5.8 STC15W408AS系列单片机的管脚说明

管脚	管脚编号				说明	
	SOP28 TSSOP28 SKDIP28	QFN28	SOP20 DIP20 TSSOP20	SOP16 DIP16		
P1.0/ADC0/ CCP1	3	27	19	15	P1.0	标准I/O口 PORT1[0]
					ADC0	ADC 输入通道-0
					CCP1	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-1
P1.1/ADC1/ CCP0	4	28	20	16	P1.1	标准I/O口 PORT1[1]
					ADC1	ADC 输入通道-1
					CCP0	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-0
P1.2/ADC2/SS/ ECI/CMPO	5	1	1	1	P1.2	标准I/O口 PORT1[2]
					ADC2	ADC 输入通道-2
					SS	SPI同步串行接口的从机选择信号
					ECI	CCP / PCA计数器的外部脉冲输入脚
					CMPO	比较器的比较结果输出管脚
P1.3/ADC3/ MOSI	6	2	2	2	P1.3	标准I/O口 PORT1[3]
					ADC3	ADC 输入通道-3
					MOSI	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P1.4/ADC4/ MISO	7	3	3	3	P1.4	标准I/O口 PORT1[4]
					ADC4	ADC 输入通道-4
					MISO	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P1.5/ADC5/ SCLK	8	4	4	4	P1.5	标准I/O口 PORT1[5]
					ADC5	ADC 输入通道-5
					SCLK	SPI同步串行接口的时钟信号
P1.6/ADC6/ RxD_3/ XTAL2/ MCLKO_2	9	5	5		P1.6	标准I/O口 PORT1[6]
					ADC6	ADC 输入通道-6
					RxD_3	串口数据接收端
					XTAL2	内部时钟电路反相放大器的输出端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚可浮空，此时XTAL2实际将XTAL1输入的时钟进行输出。
					MCLKO_2	主时钟输出;输出的频率可为MCLK/1, MCLK/2, MCLK/4. 主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK指主时钟频率。
P1.7/ADC7/ TxD_3/XTAL1	10	6	6		P1.7	标准I/O口 PORT1[7]
					ADC7	ADC 输入通道-7
					TxD_3	串口数据发送端
					XTAL1	内部时钟电路反相放大器输入端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚是外部时钟源的输入端。

STC MCU Limited

管脚	管脚编号				说明	
	SOP28 TSSOP28 SKDIP28	QFN28	SOP20 DIP20 TSSOP20	SOP16 DIP16		
P2.0/ RSTOUT_LOW	23	19			P2.0	标准I/O口 PORT2[0]
					RSTOUT_LOW	上电后,输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平,如果要读外部状态,可将该口先置高后再读
P2.1/SCLK_2	24	20			P2.1	标准I/O口 PORT2[1]
					SCLK_2	SPI同步串行接口的时钟信号
P2.2/MISO_2	25	21			P2.2	标准I/O口 PORT2[2]
					MISO_2	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P2.3/MOSI_2	26	22			P2.3	标准I/O口 PORT2[3]
					MOSI_2	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P2.4/ECI_3/ SS_2	27	23			P2.4	标准I/O口 PORT2[4]
					ECI_3	CCP / PCA计数器的外部脉冲输入脚
					SS_2	SPI同步串行接口的从机选择信号
P2.5/CCP0_3	28	24			P2.5	标准I/O口 PORT2[5]
					CCP0_3	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-0
P2.6/CCP1_3	1	25			P2.6	标准I/O口 PORT2[6]
					CCP1_3	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-1
P2.7/CCP2_3	2	26			P2.7	标准I/O口 PORT2[7]
					CCP2_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
P3.0/RxD/$\overline{\text{INT4}}$ /T2CLKO	15	11	11	9	P3.0	标准I/O口 PORT3[0]
					RxD	串口数据接收端
					$\overline{\text{INT4}}$	外部中断4,只能下降沿中断, $\overline{\text{INT4}}$支持掉电唤醒
					T2CLKO	T2的时钟输出 可通过设置INT_CLKO[2]位/T2CLKO将该管脚配置为T2CLKO
P3.1/TxD/T2	16	12	12	10	P3.1	标准I/O口 PORT3[1]
					TxD	串口数据发送端
					T2	定时器/计数器2的外部输入
P3.2/INT0	17	13	13	11	P3.2	标准I/O口 PORT3[2]
					INT0	外部中断0，既可上升沿中断也可下降沿中断.如果IT0(TCON.0)被置为1,INT0管脚仅为下降沿中断。如果IT0(TCON.0)被清0，INT0管脚既支持上升沿中断也支持下降沿中断。 INT0支持掉电唤醒。

STC MCU Limited

管脚	管脚编号				说明	
	SOP28 TSSOP28 SKDIP28	QFN28	SOP20 DIP20 TSSOP20	SOP16 DIP16		
P3.3/INT1	18	14	14	12	P3.3	标准I/O口 PORT3[3]
					INT1	外部中断1，既可上升沿中断也可下降沿中断. 如果IT1(TCON.2)被置为1, INT1管脚仅为下降沿中断。如果IT1(TCON.2)被清0，INT1管脚既支持上升沿中断也支持下降沿中断。 INT1支持掉电唤醒。
P3.4/T0/ECI_2	19	15	15		P3.4	标准I/O口 PORT3[4]
					T0	定时器/计数器0的外部输入
					ECI_2	CCP/PCA计数器的外部脉冲输入脚
P3.5/T0CLKO/ CCP0_2	20	16	16		P3.5	标准I/O口 PORT3[5]
					T0CLKO	定时器/计数器0的时钟输出 可通过设置INT_CLKO[0]位/T0CLKO将该管脚配置为T0CLKO，也可对T0脚的外部时钟输入进行分频输出
					CCP0_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P3.6/$\overline{\text{INT2}}$/RxD_2 /CCP1_2	21	17	17	13	P3.6	标准I/O口 PORT3[6]
					$\overline{\text{INT2}}$	外部中断2, 只能下降沿中断 $\overline{\text{INT2}}$支持掉电唤醒
					RxD_2	串口数据接收端
					CCP1_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
P3.7/$\overline{\text{INT3}}$/TxD_2/ CCP2/CCP2_2	22	18	18	14	P3.7	标准I/O口 PORT3[7]
					$\overline{\text{INT3}}$	外部中断3, 只能下降沿中断 $\overline{\text{INT3}}$支持掉电唤醒
					TxD_2	串口数据发送端
					CCP2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
					CCP2_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
P5.4/RST/ MCLKO/CMP-	11	7	7	5	P5.4	标准I/O口 PORT5[4]
					RST	复位脚(高电平复位)
					MCLKO	主时钟输出;输出的频率可为MCLK/1, MCLK/2, MCLK/4 (MCLK是指主时钟频率)。 主时钟可以是内部R/C时钟, 也可是外部输入的时钟或外部晶体振荡产生的时钟, MCLK指主时钟频率。
					CMP-	比较器负极输入端
P5.5/CMP+	13	9	9	7	P5.5	标准I/O口 PORT5[5]
					CMP+	比较器正极输入端
Vcc	12	8	8	6	电源正极	
Gnd	14	10	10	8	电源负极，接地	

STC MCU Limited

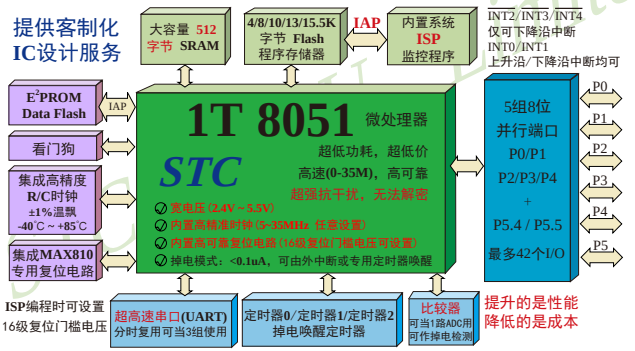
1.6 STC15W408S系列单片机总体介绍(大批量现货供应中)

1.6.1 STC15W408S系列单片机简介

STC15W408S系列单片机是STC生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/超强抗干扰的新一代8051单片机，采用STC第九代加密技术，无法解密，指令代码完全兼容传统8051，但速度快8-12倍。内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP编程时16级复位门槛电压可选)。3个16位定时器/计数器，双数据指针，1组高速异步串行通信口(UART，可在3组管脚之间进行切换，分时复用可作3组串口使用)，1组高速同步串行通信端口SPI，针对串行口通信/电机控制/强干扰场合。内置比较器，功能更强大。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通8051快8-12倍
2. 工作电压：2.4V - 5.5V
3. 4K / 8K / 10K / 13K / 15.5K字节片内Flash程序存储器，擦写次数10万次以上
4. 片内集成512字节的SRAM，包括常规的256字节RAM <idata> 和内部扩展的256字节 XRAM <xdata>
5. 有片内EEPROM功能，擦写次数10万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 内部高可靠复位，ISP编程时16级复位门槛电压可选，可彻底省掉外部复位电路
8. 内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时内部时钟从5MHz~35MHz可设(5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz)
9. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号

10. 工作频率范围：5MHz～35MHz，相当于普通8051的60MHz～420MHz

11.一组高速异步串行通信端口UART，可在3组管脚之间切换，分时复用可当3组串口使用：
串行口(RxD/P3.0, TxD/P3.1)可以切换到(RxD_2/P3.6, TxD_2/P3.7)，
还可以切换到(RxD_3/P1.6, TxD_3/P1.7).

12.一组高速同步串行通信端口SPI

13.支持程序加密后传输，防拦截

14.支持RS485下载

15.低功耗设计：低速模式，空闲模式，掉电模式/停机模式。

16. 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。

17.可将掉电模式/停机模式唤醒的资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可),$\overline{INT2}$/P3.6, $\overline{INT3}$/P3.7, $\overline{INT4}$/P3.0($\overline{INT2}$/$\overline{INT3}$/$\overline{INT4}$仅可下降沿中断)；管脚RxD(可在RxD/P3.0、RxD_2/P3.6和RxD_3/P1.6之间切换)；管脚T0/T1/T2(下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；内部低功耗掉电唤醒专用定时器。

18. 共3个定时器/计数器，3个16位可重装载的定时器/计数器(T0/T1/T2，其中T0/T1兼容普通8051的定时器/计数器)，并都可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4）。

19. 可编程时钟输出功能(对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出)：
由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz.;
而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz.

① T0在P3.5/T0CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；

② T1在P3.4/T1CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出)；

③ T2在P3.0/T2CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；

以上3个定时器/计数器均可1~65536级分频输出。

④ 主时钟在P5.4/MCLKO或P1.6/MCLKO_2对外输出时钟，并可如下分频MCLK/1, MCLK/2, MCLK/4.

STC15W408S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。
MCLK是指主时钟频率，MCLKO是指主时钟输出。

STC MCU Limited

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/P1.6口对外输出主时钟。

20. 比较器，可当1路ADC使用，并可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出（可设置极性），也支持外部管脚CMP+与内部参考电压进行比较
21. 硬件看门狗(WDT)
22. 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
23. 通用I/O口(42/38/30/26个)，复位后为： 准双向口/弱上拉（普通8051传统I/O口）

    可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏

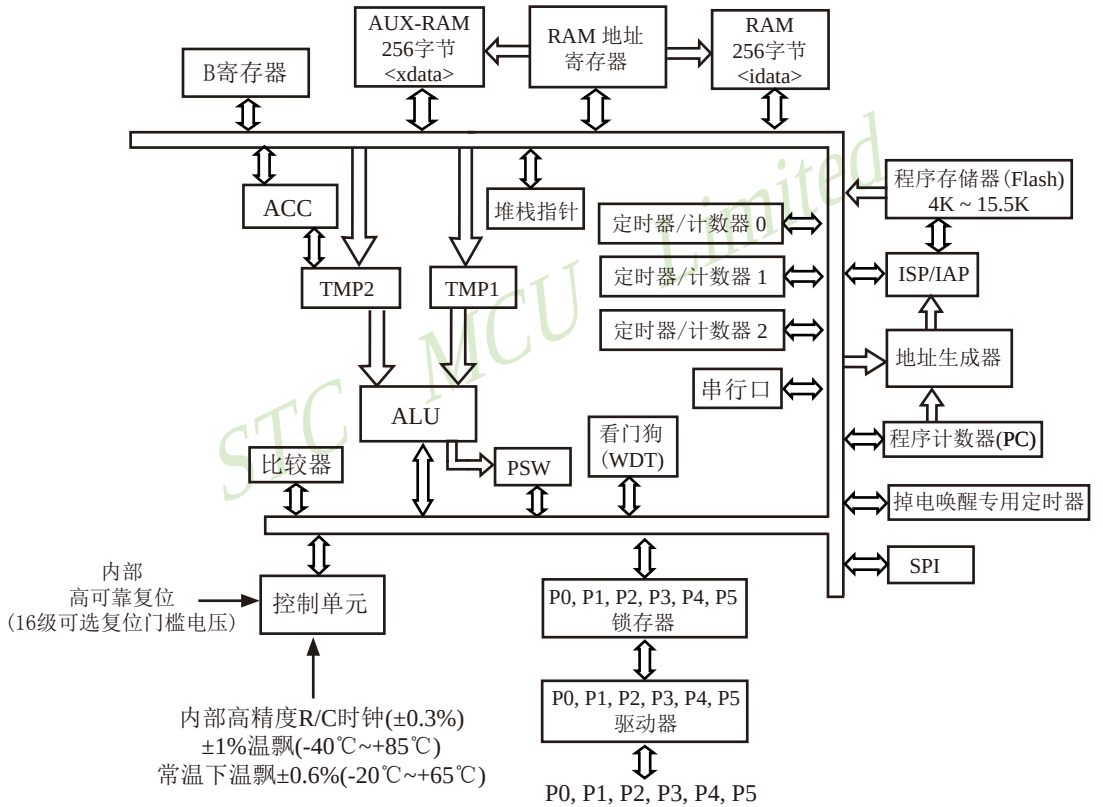
    每个I/O口驱动能力均可达到20mA，但整个芯片电流最大不要超过90mA.

    如果I/O口不够用，可外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口。
24. 封装：LQFP44(12mm x 12mm), LQFP32(9mm x 9mm), SOP28, SKDIP28, PDIP40.
25. 全部175℃八小时高温烘烤，高品质制造保证
26. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

STC MCU Limited

1.6.2 STC15W408S系列单片机的内部结构图

STC15W408S系列单片机的内部结构框图如下图所示。STC15W408S系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、内部掉电唤醒专用定时器、I/O口、比较器、看门狗、比较器，高速异步串行通信端口UART，高速同步串行端口SPI，片内高精度R/C时钟及高可靠复位等模块。STC15W408S系列单片机几乎包含了数据采集和控制中所需要的所有单元模块，可称得上是一个片上系统(SysTem Chip或SysTem on Chip,简写为STC,这是宏晶科技STC名称的由来)。



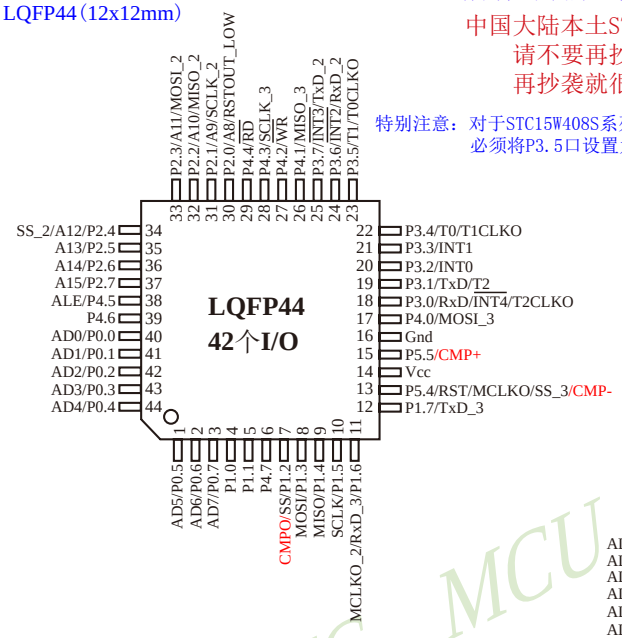
STC15W408S系列内部结构框图

1.6.3 STC15W408S系列单片机管脚图

所有封装形式均满足欧盟RoHS要求

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

LQFP44（12x12mm）



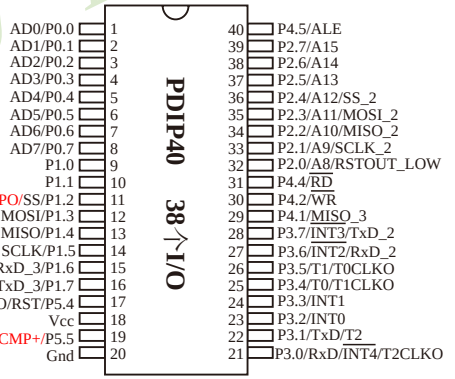
特别注意：对于STC15W408S系列单片机，若要使用T0CLKO时钟输出功能，必须将P3.5口设置为强推挽输出模式。

T0CLKO是指定时器/计数器0的可编程时钟输出（对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出）；

T1CLKO是指定时器/计数器1的可编程时钟输出（对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出）；

T2CLKO是指定时器/计数器2的可编程时钟输出（对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出）；

T0CLKO/T1CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T1/T2的时钟输入进行时钟分频输出，作分频器使用。



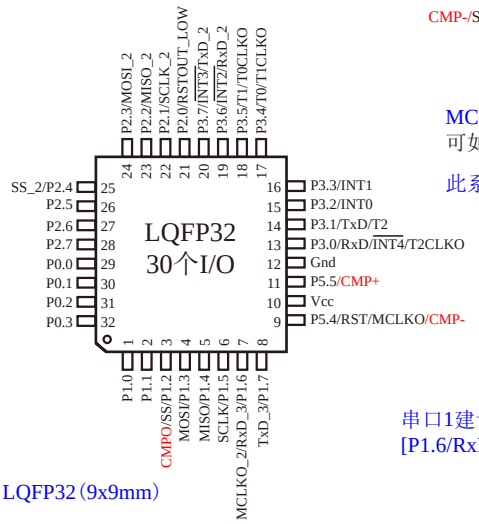
特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用。

MCLKO_2与MCLKO均是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4.

此系列的主时钟只可以是内部R/C时钟。MCLK是指主时钟频率。

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；



LQFP32（9x9mm）

串口1建议放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]上。

STC MCU Limited

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

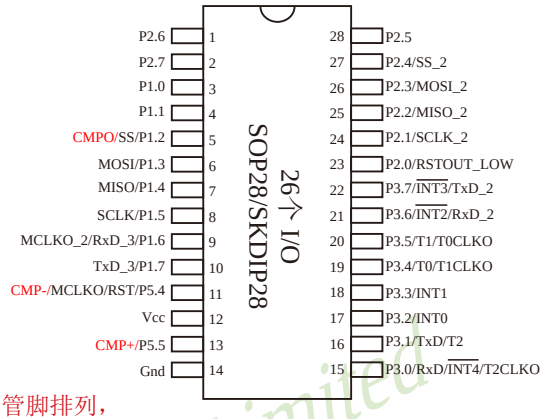
对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；

MCLKO_2与MCLKO均是指主时钟输出，
主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4.

此系列的主时钟只可以是内部R/C时钟。
MCLK是指主时钟频率。

串口1建议放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]上。

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...



特别注意：对于STC15W408S系列单片机，若要使用T0CLKO时钟输出功能，必须将P3.5口设置为强推挽输出模式。

T0CLKO是指定时器/计数器0的可编程时钟输出(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出);
T1CLKO是指定时器/计数器1的可编程时钟输出(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出);
T2CLKO是指定时器/计数器2的可编程时钟输出(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出);

T0CLKO/T1CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T1/T2的时钟输入进行时钟分频输出，作分频器使用。

STC MCU Limited

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	00xx,0000
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

串口1/S1可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD, P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2, P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3, P1.7/TxD_3] 串口1在P1口时要使用内部时钟
1	1	无效

串口1建议放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]上。

建议用户在程序中将[S1_S1, S1_S0]的值设置为[0, 1]或[1, 0]，进而将串口1放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1]上

SPI可在3个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI可在P1/P2/P4之间来回切换
0	0	SPI在[P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK]
0	1	SPI在[P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2]
1	0	SPI在[P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3]
1	1	无效

DPS：DPTR registers select bit. DPTR 寄存器选择位

0: DPTR0 is selected　　DPTR0被选择

1: DPTR1 is selected　　DPTR1被选择

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， 输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， 输出时钟频率 = MCLK / 4

STC15W408S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟，MCLK是指主时钟频率。
STC15W408S系列单片机在MCLKO/P5.4口或MCLKO_2/P1.6口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列)均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/P1.6口对外输出主时钟。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	00xx,0000
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

STC15W408S系列单片机通过CLK_DIV.3/MCLKO_2位来选择是在MCLKO/P5.4口对外输出主时钟，还是在MCLKO_2/P1.6口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位

0：在MCLKO/P5.4口对外输出主时钟；

1：在MCLKO_2/P1.6口对外输出主时钟；

STC15W408S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

串口1的RxD管脚和TxD管脚可以在3组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1]；
[RxD_2/P3.6, TxD_2/P3.7]；
[RxD_3/P1.6, TxD_3/P1.7].

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给CPU、串行口、SPI、定时器的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

STC15W408S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。

STC MCU Limited

1.6.4 STC15W408S系列单片机选型价格一览表

型号	工作电压（V）	Flash程序存储器(byte)	大容量SRAM字节	串行口并可掉电唤醒	SPI	普通定时器计数器T0-T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器(可当1路A/D使用,可作外部掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	所有封装 LQFP44/PDIP40 LQFP32 SOP28/SKDIP28 部分封装价格(RMB ¥) LQFP44	SOP28
STC15W408S系列单片机选型价格一览表																						
STC15W404S	5.5-2.4	4K	512	1	有	3	–	有	5	–	有	2	9K	有	有	16级	有	是	有	是	¥2.5	¥2.3
STC15W408S	5.5-2.4	8K	512	1	有	3	–	有	5	–	有	2	5K	有	有	16级	有	是	有	是	¥2.7	¥2.5
STC15W410S	5.5-2.4	10K	512	1	有	3	–	有	5	–	有	2	3K	有	有	16级	有	是	有	是	¥2.8	
IAP15W413S	5.5-2.4	13K	512	1	有	3	–	有	5	–	有	2	IAP	有	有	16级	有	是	有	是	¥2.8 用户可将用户程序区的程序FLASH当EEPROM使用	¥2.6
IRC15W415S 固定使用内部24MHz时钟	5.5-2.4	15.5K	512	1	有	3	–	有	5	–	有	2	IAP	有	有	固定	有	是	无	否	¥2.8 用户可将用户程序区的程序FLASH当EEPROM使用	¥2.6

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

上表中IRC15W415S型号的单片机固定使用内部24MHz时钟，且其内部复位门槛电压固定，同时不支持"程序加密后传输"功能，其P5.4不可当复位管脚RST使用，且P3.2/P3.3与下载无关。

以上单价为10K起订
量小每片需加0.1元
以上价格运费由客户承担,零售10片起
如对价格不满，可来电要求降价

提供客制化IC服务

如果用户要用40-pin及以上的单片机，建议选用LQFP44的封装，但PDIP40封装仍正常供货；
如果用户要用32-pin单片机，建议用户选用LQFP32封装；
如果用户要用28-pin单片机，建议用户选用SOP28封装。

特别注意：对于STC15W408S系列单片机，若要使用T0CLKO时钟输出功能，必须将P3.5口设置为强推挽输出模式。

总结：STC15W408S系列单片机（含IRC15W415S）有：3普通定时器/计数器(这3个普通定时器/计数器是指：T0、T1和T2)；掉电唤醒定时器；5个支持掉电唤醒的外部中断INT0/INT1/INT2/INT3/INT4；1组高速异步串行通信端口；1组高速同步串行通信端口SPI；1个比较器；2个数据指针DPTR等功能。STC15W408S系列单片机有外部数据总线。

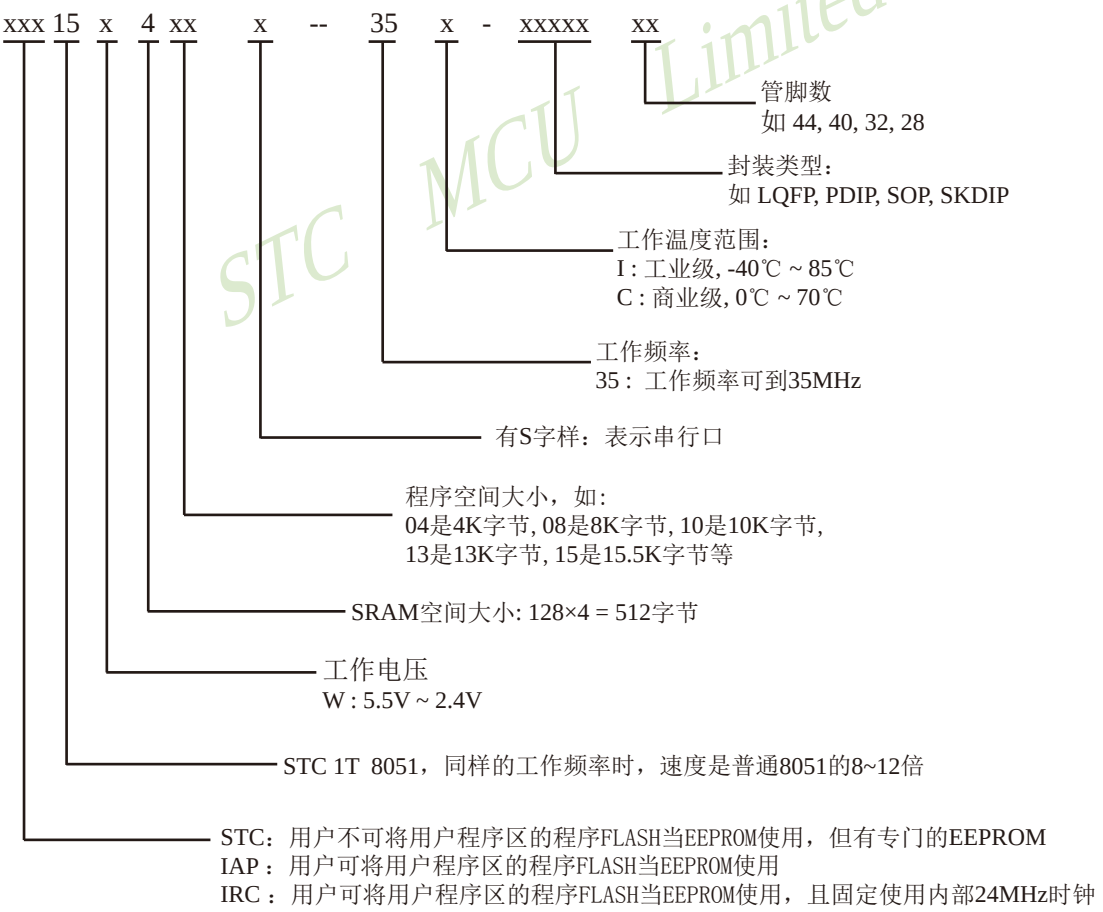
因为程序区的最后7个字节单元被强制性的放入全球唯一ID号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少7个字节。

STC MCU Limited

1.6.5 STC15W408S系列单片机封装价格一览表

型号	工作频率 (MHz)	工作温度 (I — 工业级)	所有封装价格(RMB ¥) LQFP44 / PDIP40 / LQFP32 / SOP28 / SKDIP28 (此系列暂未生产PLCC44和SOP32封装)				
			LQFP44	PDIP40	LQFP32	SOP28	SKDIP28
STC15W404S	35	-40℃ ~ +85℃	¥2.5	¥3.0	¥2.4	¥2.3	¥2.5
STC15W408S	35	-40℃ ~ +85℃	¥2.7	¥3.0	¥2.6	¥2.5	¥2.7
STC15W410S	35	-40℃ ~ +85℃	¥2.8				
IAP15W413S	35	-40℃ ~ +85℃	¥2.8	¥3.3	¥2.7	¥2.6	¥2.8
IRC15W415S	35	-40℃ ~ +85℃	¥2.8	¥3.3	¥2.7	¥2.6	¥2.8

1.6.6 STC15W408S系列单片机命名规则



STC MCU Limited

命名举例：

（1）STC15W404S- 35I - SOP28 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为512字节，程序空间大小为4K，有1组高速异步串行通信端口UART，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为28。

（2）STC15W408S- 35I - LQFP44 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为512字节，程序空间大小为8K，有1组高速异步串行通信端口UART，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

（3）STC15W410S- 35I - SKDIP28 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为512字节，程序空间大小为10K，有1组高速异步串行通信端口UART，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SKDIP封装，管脚数为28。

（4）IAP15W413S- 35I - PDIP40 表示：
用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为512字节，程序空间大小为13K，有1组高速异步串行通信端口UART，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为PDIP封装，管脚数为40。

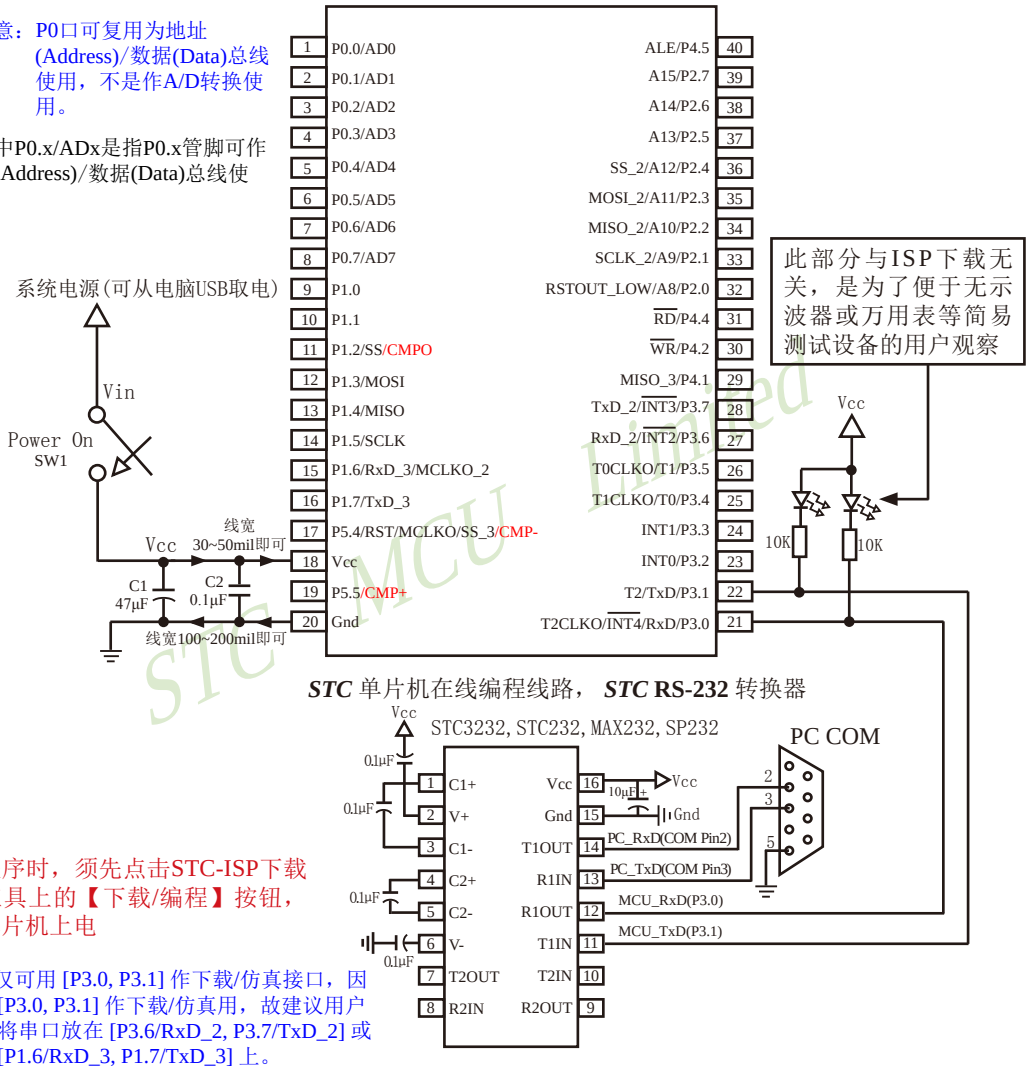
STC MCU Limited

1.6.7 STC15W408S系列单片机在系统可编程(ISP)典型应用线路图

1.6.7.1 利用RS-232转换器的ISP下载编程典型应用线路图

特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。

管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用。



STC 单片机在线编程线路， ***STC*** **RS-232** 转换器

烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

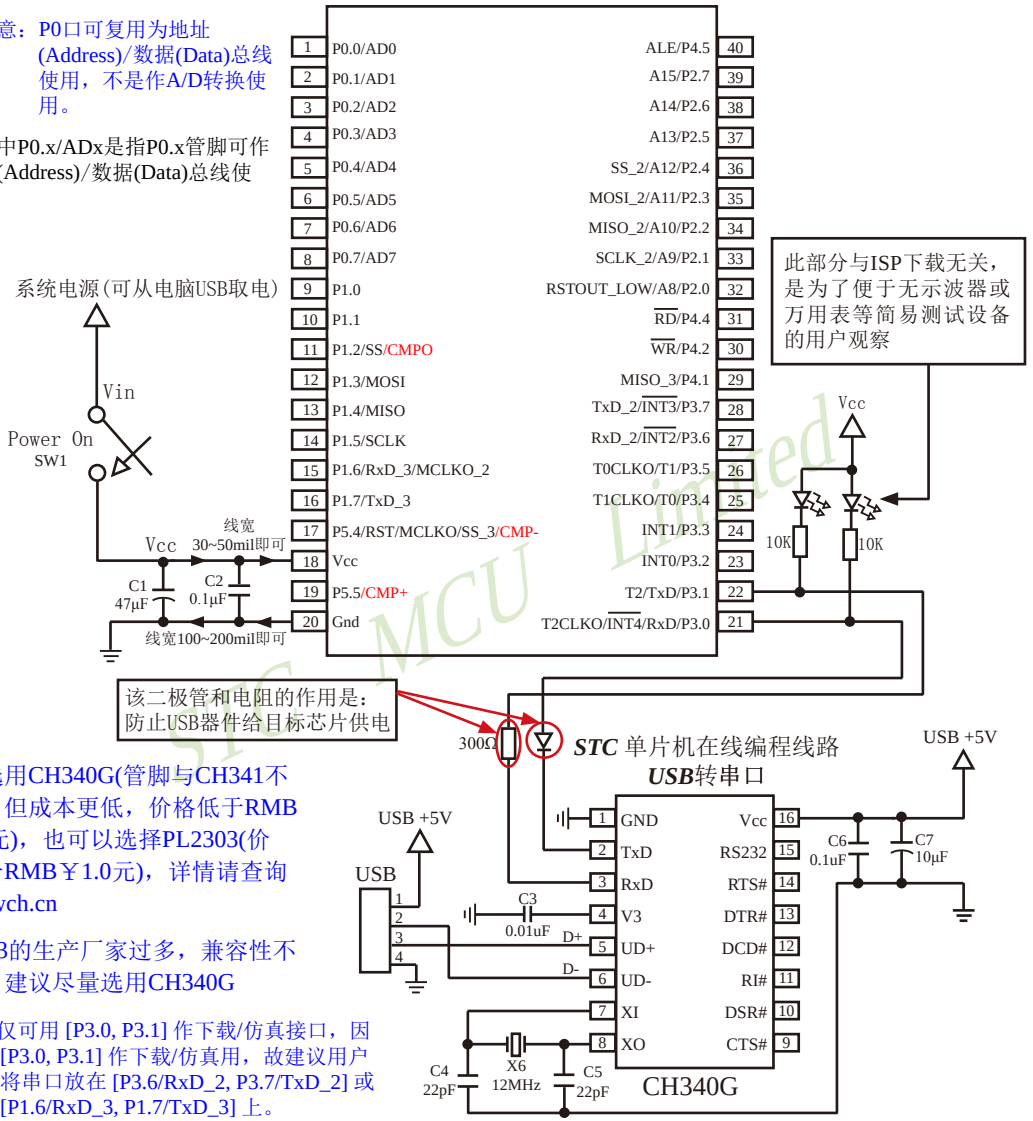
建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

STC MCU limited

1.6.7.2 利用USB转串口的ISP下载编程典型应用线路图

特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。

管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用。



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.6.8 STC15W408S系列单片机的管脚说明

管脚	管脚编号						说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P0.0/AD0	40	2	1	1	29	-	P0：P0口既可作为输入/输出口，也可作为地址/数据复用总线使用。当P0口作为输入/输出口时，P0可以由软件配置成准双向口/弱上拉、推挽输出/强上拉、仅为输入(高阻)及开漏输出等4种工作类型之一，上电复位后为准双向口/弱上拉模式。当P0作为地址/数据复用总线使用时，是低8位地址线[A0~A7]及数据线的[D0~D7]。	
P0.1/AD1	41	3	2	2	30	-		
P0.2/AD2	42	4	3	3	31	-		
P0.3/AD3	43	5	4	4	32	-		
P0.4/AD4	44	6	5	-	-	-		
P0.5/AD5	1	7	6	-	-	-		
P0.6/AD5	2	8	7	-	-	-		
P0.7/AD7	3	9	8	-	-	-		
P1.0	4	10	9	5	1	3	标准I/O口 PORT1[0]	
P1.1	5	11	10	6	2	4	标准I/O口 PORT1[1]	
P1.2/SS/CMPO	7	13	11	7	3	5	P1.2	标准I/O口 PORT1[2]
							SS	SPI同步串行接口的从机选择信号
							CMPO	比较器的比较结果输出管脚
P1.3/MOSI	8	14	12	8	4	6	P1.3	标准I/O口 PORT1[3]
							MOSI	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P1.4/MISO	9	15	13	9	5	7	P1.4	标准I/O口 PORT1[4]
							MISO	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P1.5/SCLK	10	16	14	10	6	8	P1.5	标准I/O口 PORT1[5]
							SCLK	SPI同步串行接口的时钟信号
P1.6/RxD_3/MCLKO_2	11	17	15	11	7	9	P1.6	标准I/O口 PORT1[6]
							RxD_3	串口1数据接收端
							MCLKO_2	主时钟输出;输出的频率可为MCLK/1, MCLK/2, MCLK/4. 此系列的主时钟只可以是内部R/C时钟，MCLK指主时钟频率。
P1.7/TxD_3	12	18	16	12	8	10	P1.7	标准I/O口 PORT1[7]
							TxD_3	串口1数据发送端
P2.0/A8/RSTOUT_LOW	30	36	32	25	21	23	P2.0	标准I/O口 PORT2[0]
							A8	地址总线第8位 — A8
							RSTOUT_LOW	上电后,输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P2.1/A9/SCLK_2	31	37	33	26	22	24	P2.1	标准I/O口 PORT2[1]
							A9	地址总线第9位 — A9
							SCLK_2	SPI同步串行接口的时钟信号
P2.2/A10/MISO_2	32	38	34	27	23	25	P2.2	标准I/O口 PORT2[2]
							A10	地址总线第10位 — A10
							MISO_2	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)

STC MCU Limited

管脚	管脚编号						说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P2.3/A11/ MOSI_2	33	39	35	28	24	26	P2.3	标准I/O口 PORT2[3]
							A11	地址总线第11位 — A11
							MOSI_2	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P2.4/A12/ SS_2	34	40	36	29	25	27	P2.4	标准I/O口 PORT2[4]
							A12	地址总线第12位 — A12
							SS_2	SPI同步串行接口的从机选择信号
P2.5/A13	35	41	37	30	26	28	P2.5	标准I/O口 PORT2[5]
							A13	地址总线第13位 — A13
P2.6/A14	36	42	38	31	27	1	P2.6	标准I/O口 PORT2[6]
							A14	地址总线第14位 — A14
P2.7/A15	37	43	39	32	28	2	P2.7	标准I/O口 PORT2[7]
							A15	地址总线第15位 — A15
P3.0/RxD/ $\overline{\text{INT4}}$ /T2CLKO	18	24	21	17	13	15	P3.0	标准I/O口 PORT3[0]
							RxD	串口1数据接收端
							$\overline{\text{INT4}}$	外部中断4, 只能下降沿中断, /INT4支持掉电唤醒
							T2CLKO	T2的时钟输出 可通过设置INT_CLKO[2]位/T2CLKO将该管脚配置为T2CLKO
P3.1/TxD/ T2	19	25	22	18	14	16	P3.1	标准I/O口 PORT3[1]
							TxD	串口1数据发送端
							T2	定时器/计数器2的外部输入
P3.2/INT0	20	26	23	19	15	17	P3.2	标准I/O口 PORT3[2]
							INT0	外部中断0，既可上升沿中断也可下降沿中断. 如果IT0(TCON.0)被置为1, INT0管脚仅为下降沿中断。如果IT0(TCON.0)被清0，INT0管脚既支持上升沿中断也支持下降沿中断。INT0支持掉电唤醒。
P3.3/INT1	21	27	24	20	16	18	P3.3	标准I/O口 PORT3[3]
							INT1	外部中断1，既可上升沿中断也可下降沿中断. 如果IT1(TCON.2)被置为1, INT1管脚仅为下降沿中断。如果IT1(TCON.2)被清0，INT1管脚既支持上升沿中断也支持下降沿中断。INT1支持掉电唤醒。
P3.4/T0/ T1CLKO	22	28	25	21	17	19	P3.4	标准I/O口 PORT3[4]
							T0	定时器/计数器0的外部输入
							T1CLKO	定时器/计数器1的时钟输出 可通过设置INT_CLKO[1]位/T1CLKO将该管脚配置为T1CLKO，也可对T1脚的外部时钟输入进行分频输出

STC MCU Limited

管脚	管脚编号						说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P3.5/T1/ T0CLKO	23	29	26	22	18	20	P3.5	标准I/O口 PORT3[5]
							T1	定时器/计数器1的外部输入
							T0CLKO	定时器/计数器0的时钟输出 可通过设置INT_CLKO[0]位/T0CLKO将该管脚配置为T0CLKO，也可对T0脚的外部时钟输入进行分频输出
P3.6/$\overline{\text{INT2}}$ /RxD_2	24	30	27	23	19	21	P3.6	标准I/O口 PORT3[6]
							$\overline{\text{INT2}}$	外部中断2,只能下降沿中断 支持掉电唤醒
							RxD_2	串口1数据接收端
P3.7/$\overline{\text{INT3}}$ /TxD_2	25	31	28	24	20	22	P3.7	标准I/O口 PORT3[7]
							$\overline{\text{INT3}}$	外部中断3,只能下降沿中断 支持掉电唤醒
							TxD_2	串口1数据发送端
P4.0	17	23	-	-	-	-	标准I/O口 PORT4[0]	
P4.1	26	32	29	-	-	-	标准I/O口 PORT4[1]	
P4.2/$\overline{\text{WR}}$	27	33	30	-	-	-	P4.2	标准I/O口 PORT4[2]
							$\overline{\text{WR}}$	外部数据存储器写脉冲
P4.3	28	34	-	-	-	-	标准I/O口 PORT4[3]	
P4.4/$\overline{\text{RD}}$	29	35	31	-	-	-	P4.4	标准I/O口 PORT4[4]
							$\overline{\text{RD}}$	外部数据存储器读脉冲
P4.5/ALE	38	44	40	-	-	-	P4.5	标准I/O口 PORT4[5]
							ALE	地址锁存允许
P4.6	39	1	-	-	-	-	标准I/O口 PORT4[6]	
P4.7	6	12	-	-	-	-	标准I/O口 PORT4[7]	
P5.4/RST/ MCLKO/ CMP-	13	19	17	13	9	11	P5.4	标准I/O口 PORT5[4]
							RST	复位脚(高电平复位)
							MCLKO	主时钟输出;输出的频率可为MCLK/1, MCLK/2, MCLK/4（MCLK是指主时钟频率）。 此系列的主时钟只可以是内部R/C时钟，MCLK指主时钟频率。
							CMP-	比较器负极输入端
P5.5/CMP+	15	21	19	15	11	13	P5.5	标准I/O口 PORT5[5]
							CMP+	比较器正极输入端
Vcc	14	20	18	14	10	12	电源正极	
Gnd	16	22	20	16	12	14	电源负极，接地	

STC MCU Limited

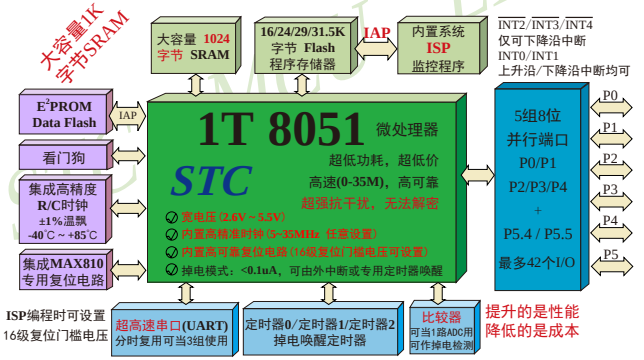
1.7 STC15W1K16S系列单片机总体介绍(大批量现货供应中)

1.7.1 STC15W1K16S系列单片机简介

STC15W1K16S系列单片机是STC生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/超强抗干扰的新一代8051单片机，采用STC第九代加密技术，无法解密，指令代码完全兼容传统8051,但速度快8-12倍。内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP编程时16级复位门槛电压可选)。3个16位定时器/计数器，1个掉电唤醒定时器，双数据指针，1组高速异步串行通信口(UART，可在3组管脚之间进行切换，分时复用可作3组串口使用)，1组高速同步串行通信端口SPI，针对串行口通信/电机控制/强干扰场合。内置比较器，功能更强大。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通8051快8-12倍
2. 工作电压：2.6V - 5.5V
3. 16K/24K/29/31.5K字节片内Flash程序存储器，擦写次数10万次以上
4. 片内集成1024字节的SRAM，包括常规的256字节RAM <idata> 和内部扩展的768字节 XRAM <xdata>
5. 有片内EEPROM功能，擦写次数10万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 内部高可靠复位，ISP编程时16级复位门槛电压可选，可彻底省掉外部复位电路
8. 内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时内部时钟从5MHz~35MHz可设(5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz)

9. 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号

10.工作频率范围：5MHz～35MHz，相当于普通8051的60MHz～420MHz

11.一组高速异步串行通信端口UART，可在3组管脚之间切换，分时复用可当3组串口使用：
串行口(RxD/P3.0, TxD/P3.1)可以切换到(RxD_2/P3.6, TxD_2/P3.7)，
还可以切换到(RxD_3/P1.6, TxD_3/P1.7)；

12.一组高速同步串行通信端口SPI

13.支持程序加密后传输，防拦截

14.支持RS485下载

15.低功耗设计：低速模式，空闲模式，掉电模式/停机模式。

16.可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。

17.可将掉电模式/停机模式唤醒的资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0($\overline{\text{INT2}}$ / $\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚RxD(可在RxD/P3.0、RxD_2/P3.6和RxD_3/P1.6之间切换)；管脚T0/T1/T2(下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；内部低功耗掉电唤醒专用定时器。

18. 共3个定时器/计数器，3个16位可重装载的定时器/计数器(T0/T1/T2，其中T0/T1兼容普通8051的定时器/计数器)，并都可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4）。

19. 可编程时钟输出功能(对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出)：
由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz.；
而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz.

① T0在P3.5/T0CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；

② T1在P3.4/T1CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出)；

③ T2在P3.0/T2CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；

以上3个定时器/计数器均可1~65536级分频输出。

④ 主时钟在P5.4/MCLKO或P1.6/XTAL2/MCLKO_2对外输出时钟，并可如下分频
MCLK/1, MCLK/2, MCLK/4.

STC15W1K16S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。
MCLK是指主时钟频率，MCLKO是指主时钟输出。

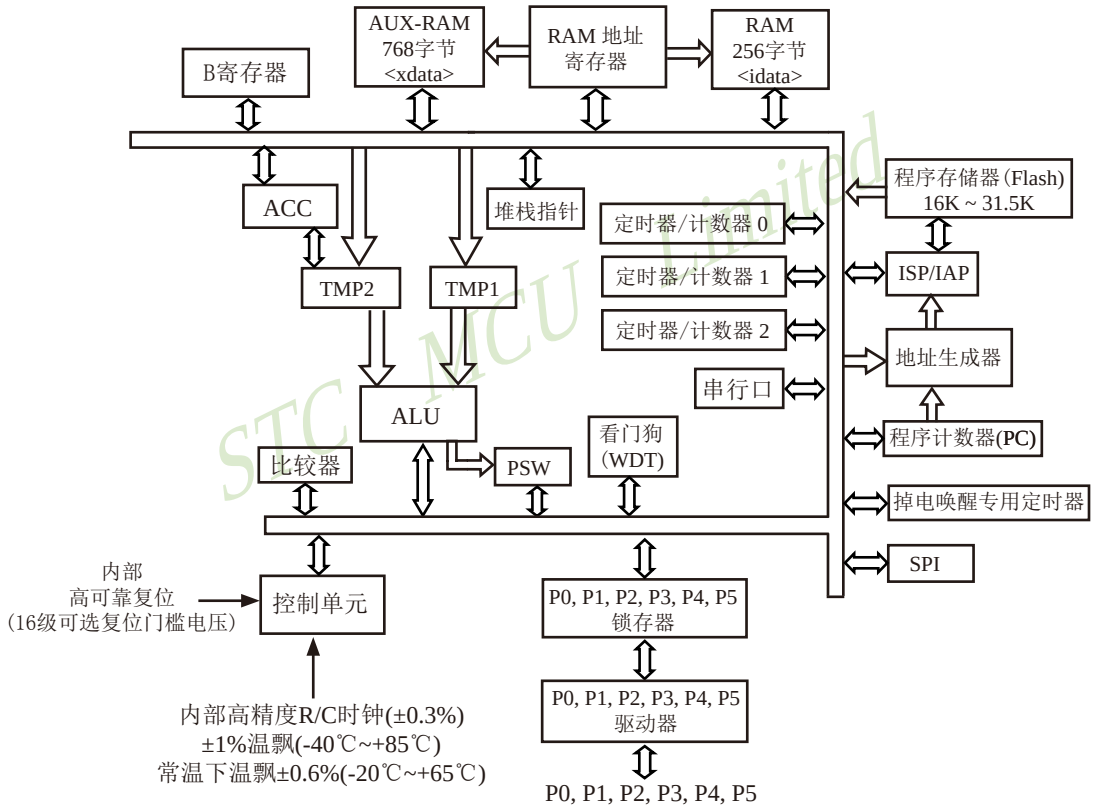
STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/P1.6口对外输出主时钟。

20. 比较器，可当1路ADC使用，并可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出（可设置极性），也支持外部管脚CMP+与内部参考电压进行比较
21. 硬件看门狗(WDT)
22. 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
23. 通用I/O口(42/38/30/26个)，复位后为：准双向口/弱上拉（普通8051传统I/O口）
    可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
    每个I/O口驱动能力均可达到20mA，但整个芯片电流最大不要超过90mA.
    如果I/O口不够用，可外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口。
24. 封装：LQFP44(12mm x 12mm), LQFP32(9mm x 9mm), SOP28, SKDIP28, PDIP40, TSSOP20(6.5mm x 6.5mm).
25. 全部175℃八小时高温烘烤，高品质制造保证
26. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

STC MCU Limited

1.7.2 STC15W1K16S系列单片机的内部结构图

STC15W1K16S系列单片机的内部结构框图如下图所示。STC15W1K16S系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、内部掉电唤醒专用定时器、I/O口、比较器、看门狗、高速异步串行通信端口UART，高速同步串行端口SPI，片内高精度R/C时钟及高可靠复位等模块。STC15W1K16S系列单片机几乎包含了数据采集和控制中所需要的所有单元模块，可称得上是一个片上系统(SysTem Chip或SysTem on Chip,简写为STC,这是宏晶科技STC名称的由来)。



STC15W1K16S系列内部结构框图

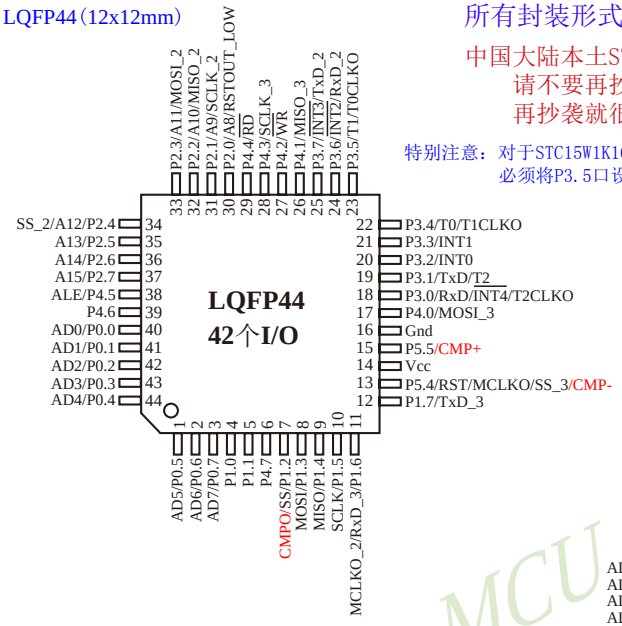
1.7.3 STC15W1K16S系列单片机管脚图

LQFP44(12x12mm)

所有封装形式均满足欧盟RoHS要求

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

特别注意：对于STC15W1K16S系列单片机，若要使用T0CLKO时钟输出功能，必须将P3.5口设置为强推挽输出模式。

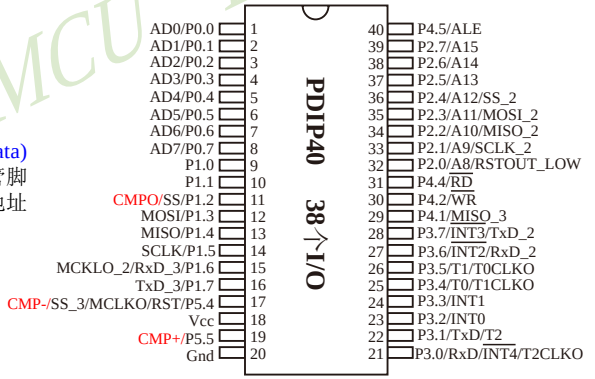


T0CLKO是指定时器/计数器0的可编程时钟输出(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出);

T1CLKO是指定时器/计数器1的可编程时钟输出(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出);

T2CLKO是指定时器/计数器2的可编程时钟输出(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出);

T0CLKO/T1CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T1/T2的时钟输入进行时钟分频输出，作分频器使用。



特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用。

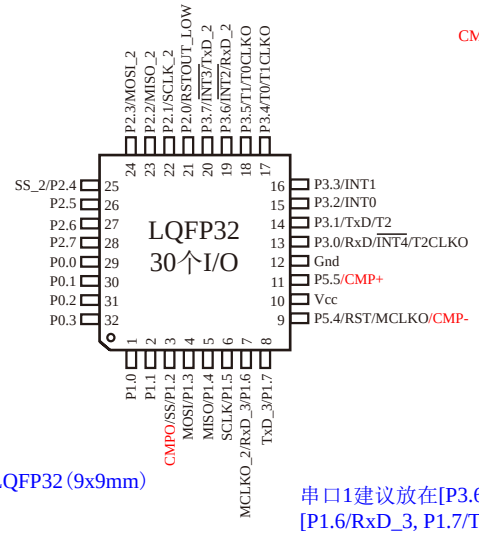
MCLKO_2与MCLKO均是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4.

此系列的主时钟只可以是内部R/C时钟。MCLK是指主时钟频率。

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz;

对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz;

LQFP32(9x9mm)



串口1建议放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3/]上。

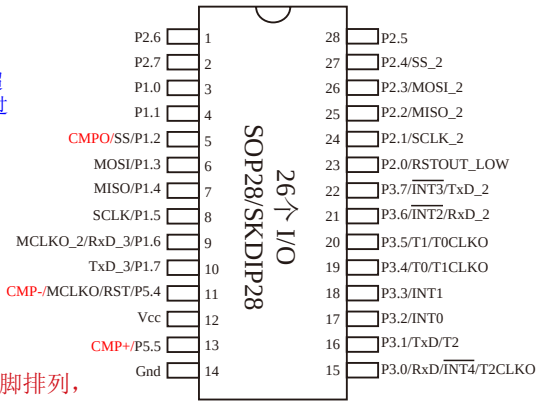
STC MCU Limited

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；

MCLKO_2与MCLKO均是指主时钟输出，
主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4.

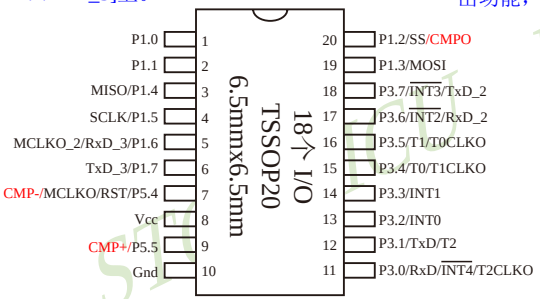
此系列的主时钟只可以是内部R/C时钟。
MCLK是指主时钟频率。



中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

串口1建议放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]上。

特别注意：对于STC15W1K16S系列单片机，若要使用T0CLKO时钟输出功能，必须将P3.5口设置为强推挽输出模式。



T0CLKO是指定时器/计数器0的可编程时钟输出(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出);
T1CLKO是指定时器/计数器1的可编程时钟输出(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出);
T2CLKO是指定时器/计数器2的可编程时钟输出(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出);

T0CLKO/T1CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T1/T2的时钟输入进行时钟分频输出，作分频器使用。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	00xx,0000
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

串口1/S1可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD, P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2, P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3, P1.7/TxD_3] 串口1在P1口时要使用内部时钟
1	1	无效

串口1建议放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]上。

建议用户在程序中将[S1_S1, S1_S0]的值设置为[0, 1]或[1, 0]，进而将串口1放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1]上

SPI可在3个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI可在P1/P2/P4之间来回切换
0	0	SPI在[P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK]
0	1	SPI在[P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2]
1	0	SPI在[P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3]
1	1	无效

DPS: DPTR registers select bit. DPTR 寄存器选择位

0: DPTR0 is selected　　DPTR0被选择

1: DPTR1 is selected　　DPTR1被选择

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， 输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， 输出时钟频率 = MCLK / 4

STC15W1K16S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。MCLK是指主时钟频率。
STC15W1K16S系列单片机在MCLKO/P5.4口或MCLKO_2/P1.6口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列)均在MCLKO/P5.4口对外输出主时钟,且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/P1.6口对外输出主时钟。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	00xx,0000
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

STC15W1K16S系列单片机通过CLK_DIV.3/MCLKO_2位来选择是在MCLKO/P5.4口对外输出主时钟，还是在MCLKO_2/P1.6口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位

0：在MCLKO/P5.4口对外输出主时钟；

1：在MCLKO_2/P1.6口对外输出主时钟；

STC15W1K16S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

串口1的RxD管脚和TxD管脚可以在3组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1];
[RxD_2/P3.6, TxD_2/P3.7];
[RxD_3/P1.6, TxD_3/P1.7].

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 （系统时钟是指对主时钟进行分频后供给CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D转换的实际工作时钟）
0	0	0	主时钟频率/1, 不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

STC15W1K16S系列单片机不支持外接外部晶体，其主时钟只可以是内部R/C时钟。

STC MCU Limited

1.7.4 STC15W1K16S系列单片机选型价格一览表

型号	工作电压（V）	Flash程序存储器(byte)	大容量SRAM字节	串行口并可掉电唤醒	SPI	普通定时器计数器T0-T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器(可当1路A/D使用,可作外部掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	所有封装 LQFP44/PDIP40 LQFP32 SOP28/SKDIP28 TSSOP20 部分封装价格(RMB ¥) LQFP44	SOP28
STC15W1K16S系列单片机选型价格一览表																						
STC15W1K16S	5.5-2.6	16K	1K	1	有	3	-	有	5	-	有	2	13K	有	有	16级	有	是	有	是	¥3.5	¥3.3
STC15W1K24S	5.5-2.6	24K	1K	1	有	3	-	有	5	-	有	2	5K	有	有	16级	有	是	有	是	¥3.8	¥3.6
IAP15W1K29S	5.5-2.6	29K	1K	1	有	3	-	有	5	-	有	2	IAP	有	有	16级	有	是	有	是	¥3.8 用户可将用户程序区的程序FLASH当EEPROM使用	¥3.6
IRC15W1K31S 固定使用内部24MHz时钟	5.5-2.6	31.5K	1K	1	有	3	-	有	5	-	有	2	IAP	有	有	固定	有	是	无	否	¥3.8 用户可将用户程序区的程序FLASH当EEPROM使用	¥3.6

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

提供客制化IC服务

如果用户要用40-pin及以上的单片机，建议选用LQFP44的封装，但PDIP40封装仍正常供货；如果用户要用32-pin单片机，建议用户选用LQFP32封装；如果用户要用28-pin单片机，建议用户选用SOP28封装。

以上单价为10K起订
量小每片需加0.1元
以上价格运费由客户承担,零售10片起
如对价格不满，可来电要求降价

TSSOP20封装仅支持IAP15W1K29S型号单片机，且IAP15W1K29S型号中封装为TSSOP20的单片机与该型号中封装为SOP28的单片机的价格相同.

因为程序区的最后7个字节单元被强制性的放入全球唯一ID号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少7个字节。

上表中IRC15W1K31S型号的单片机固定使用内部24MHz时钟，且其内部复位门槛电压固定，同时不支持"程序加密后传输"功能，其P5.4不可当复位管脚RST使用，且P3.2/P3.3与下载无关。

特别注意：对于STC15W1K16S系列单片机，若要使用T0CLKO时钟输出功能，必须将P3.5口设置为强推挽输出模式。

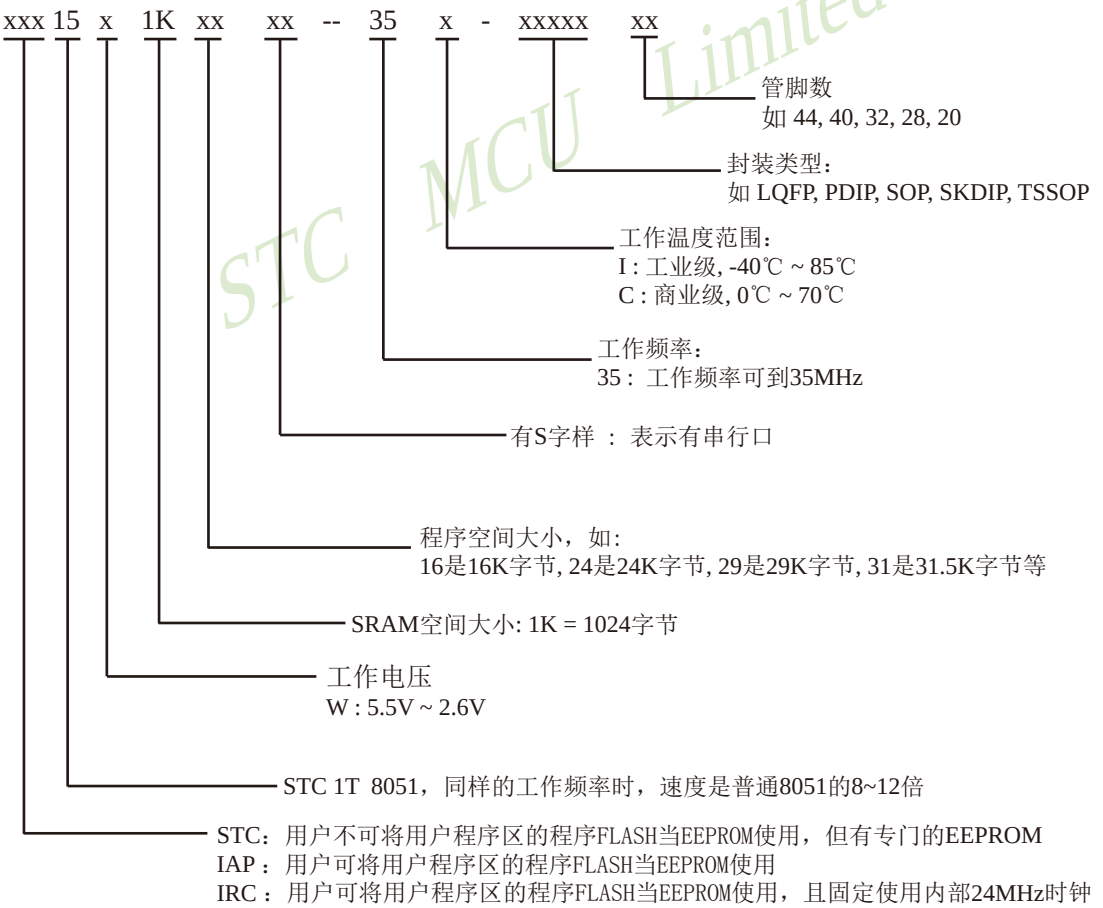
总结：STC15W1K16S系列单片机（含IRC15W1K31S）有：3普通定时器/计数器(这3个普通定时器/计数器是指：T0、T1和T2)；掉电唤醒专用定时器；5个支持掉电唤醒的外部中断INT0/INT1/$\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$；1组高速异步串行通信端口；1个比较器；2个数据指针DPTR等功能。STC15W1K16S系列单片机有外部数据总线。

STC MCU Limited

1.7.5 STC15W1K16S系列单片机封装价格一览表

型号	工作频率 (MHz)	工作温度 (I — 工业级)	所有封装价格(RMB ¥) LQFP44 / PDIP40 / LQFP32 / SOP28 / SKDIP28 / TSSOP20					
			LQFP44	PDIP40	LQFP32	SOP28	SKDIP28	TSSOP20
STC15W1K16S	35	-40℃ ~ +85℃	¥3.5	¥4.0	¥3.4	¥3.3	¥3.5	
STC15W1K24S	35	-40℃ ~ +85℃	¥3.8	¥4.3	¥3.7	¥3.6	¥3.8	¥3.6
IAP15W1K29S	35	-40℃ ~ +85℃	¥3.8	¥4.3	¥3.7	¥3.6	¥3.8	¥3.6
IRC15W1K31S	35	-40℃ ~ +85℃	¥3.8	¥4.3	¥3.7	¥3.6	¥3.8	

1.7.6 STC15W1K16S系列单片机命名规则



STC MCU Limited

命名举例：

（1）STC15W1K16S- 35I - SOP28 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.6V，SRAM空间大小为1K(1024)字节，程序空间大小为16K，有1组高速异步串行通信端口UART，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为28。

（2）STC15W1K24S- 35I - LQFP44 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.6V，SRAM空间大小为1K(1024)字节，程序空间大小为24K，有1组高速异步串行通信端口UART，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

（3）STC15W1K24S- 35I - SKDIP28 表示：
用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.6V，SRAM空间大小为1K(1024)字节，程序空间大小为24K，有1组高速异步串行通信端口UART，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SKDIP封装，管脚数为28。

（4）IAP15W1K29S- 35I - PDIP40 表示：
用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.6V，SRAM空间大小为1K(1024)字节，程序空间大小为29K，有1组高速异步串行通信端口UART，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为PDIP封装，管脚数为40。

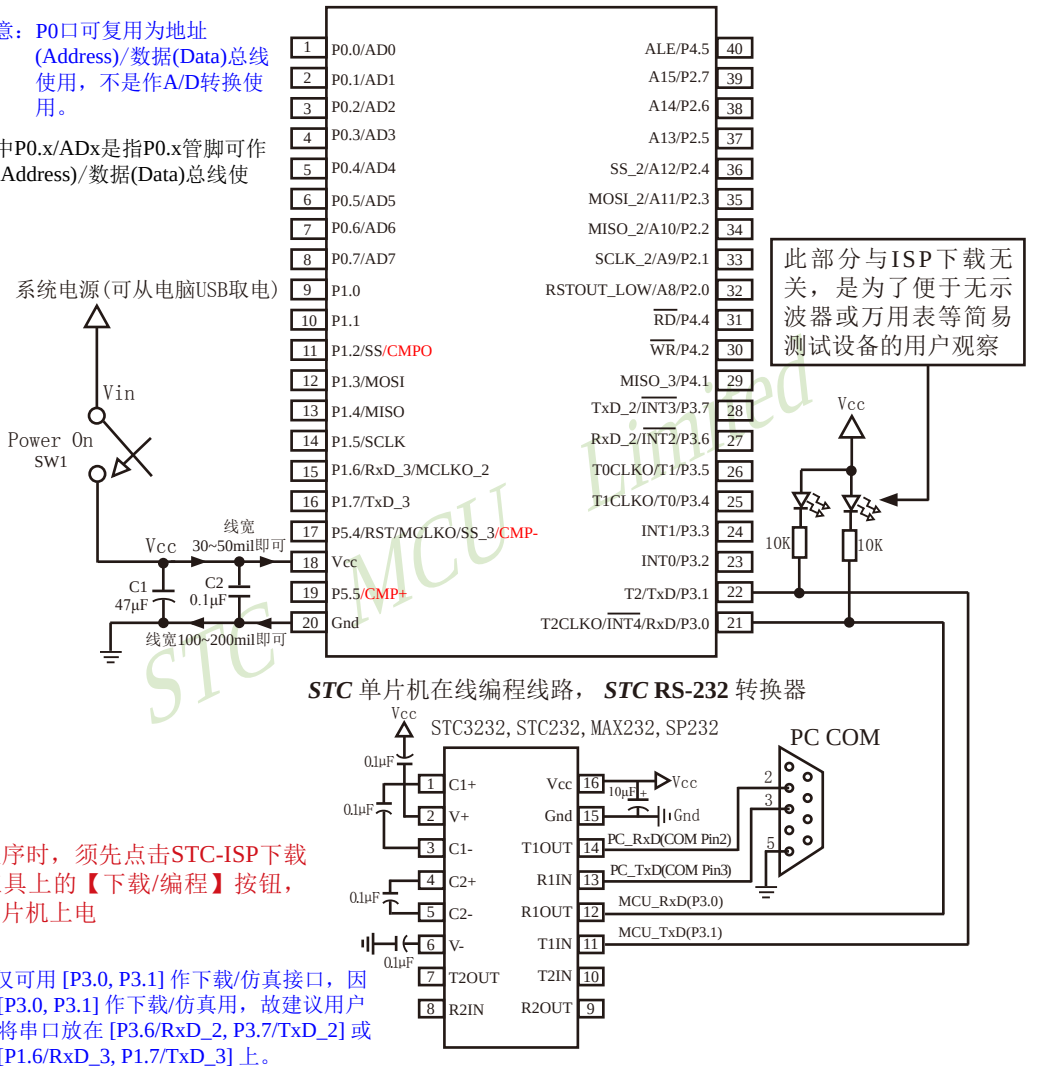
STC MCU Limited

1.7.7 STC15W1K16S系列单片机在系统可编程(ISP)典型应用线路图

1.7.7.1 利用RS-232转换器的ISP下载编程典型应用线路图

特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。

管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用。



STC 单片机在线编程线路， ***STC*** **RS-232** 转换器

烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

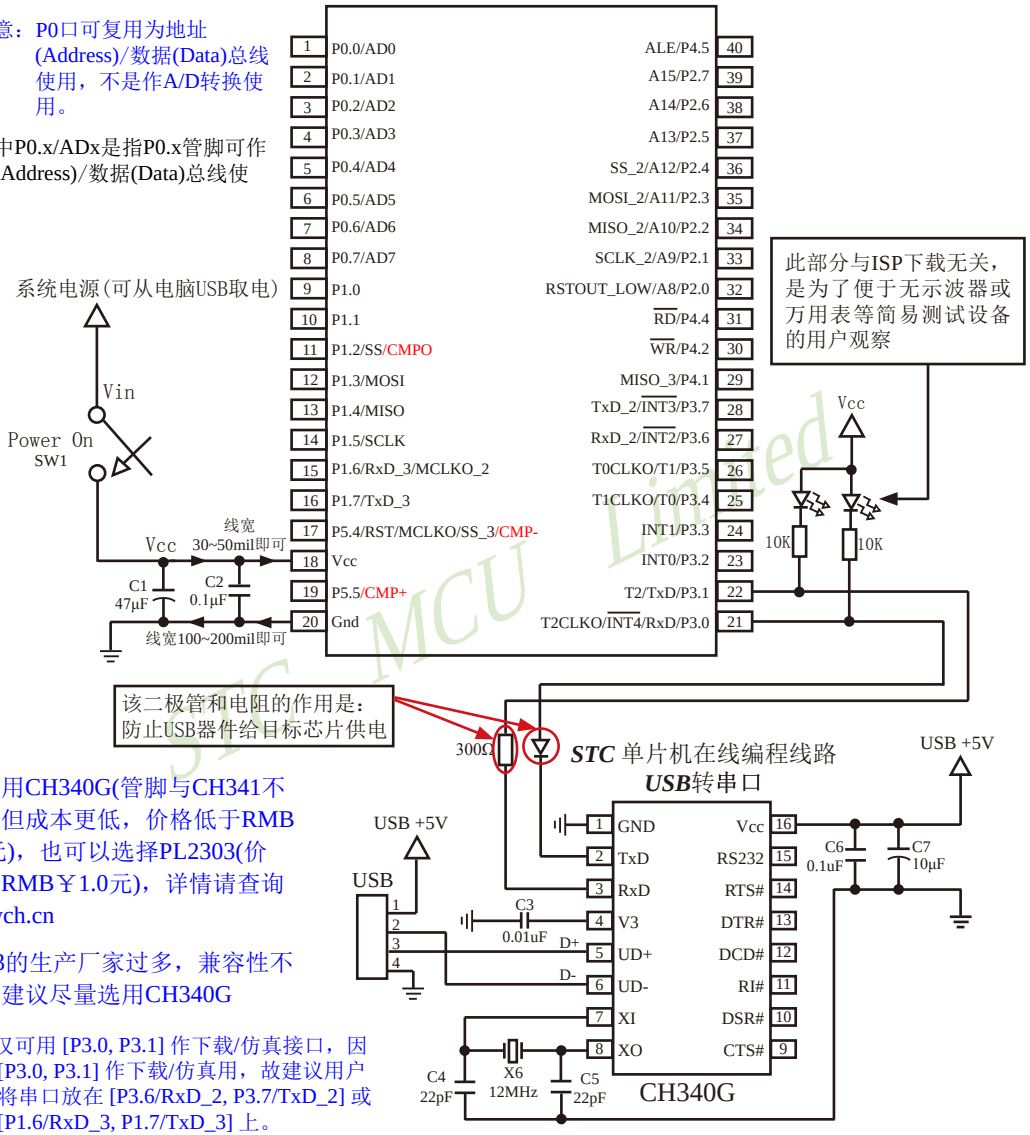
内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.7.7.2 利用USB转串口的ISP下载编程典型应用线路图

特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。

管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用。



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.7.8 STC15W1K16S系列单片机的管脚说明

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	TSSOP20		
P0.0/AD0	40	2	1	1	29	-		P0：P0口既可作为输入/输出口，也可作为地址/数据复用总线使用。当P0口作为输入/输出口时，P0可以由软件配置成准双向口/弱上拉、推挽输出/强上拉、仅为输入(高阻)及开漏输出等4种工作类型之一，上电复位后为准双向口/弱上拉模式。当P0作为地址/数据复用总线使用时，是低8位地址线[A0~A7]及数据线的[D0~D7]。	
P0.1/AD1	41	3	2	2	30	-			
P0.2/AD2	42	4	3	3	31	-			
P0.3/AD3	43	5	4	4	32	-			
P0.4/AD4	44	6	5	-	-	-			
P0.5/AD5	1	7	6	-	-	-			
P0.6/AD5	2	8	7	-	-	-			
P0.7/AD7	3	9	8	-	-	-			
P1.0	4	10	9	5	1	3	1	标准I/O口 PORT1[0]	
P1.1	5	11	10	6	2	4	2	标准I/O口 PORT1[1]	
P1.2/SS/ CMPO	7	13	11	7	3	5	20	P1.2	标准I/O口 PORT1[2]
								SS	SPI同步串行接口的从机选择信号
								CMPO	比较器的比较结果输出管脚
P1.3/MOSI	8	14	12	8	4	6	19	P1.3	标准I/O口 PORT1[3]
								MOSI	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P1.4/MISO	9	15	13	9	5	7	3	P1.4	标准I/O口 PORT1[4]
								MISO	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P1.5/SCLK	10	16	14	10	6	8	4	P1.5	标准I/O口 PORT1[5]
								SCLK	SPI同步串行接口的时钟信号
P1.6/RxD_3/ MCLKO_2	11	17	15	11	7	9	5	P1.6	标准I/O口 PORT1[6]
								RxD_3	串口1数据接收端
								MCLKO_2	主时钟输出;输出的频率可为MCLK/1, MCLK/2, MCLK/4(MCLK是指主时钟频率)。 此系列的主时钟只可以是内部R/C时钟，MCLK指主时钟频率。
P1.7/TxD_3	12	18	16	12	8	10	6	P1.7	标准I/O口 PORT1[7]
								TxD_3	串口1数据发送端
P2.0/A8/ RSTOUT_LOW	30	36	32	25	21	23		P2.0	标准I/O口 PORT2[0]
								A8	地址总线第8位 — A8
								RSTOUT_LOW	上电后,输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读

STC MCU Limited

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	TSSOP20		
P2.1/A9/ SCLK_2	31	37	33	26	22	24		P2.1	标准I/O口 PORT2[1]
								A9	地址总线第9位 — A9
								SCLK_2	SPI同步串行接口的时钟信号
P2.2/A10	32	38	34	27	23	25		P2.2	标准I/O口 PORT2[2]
								A10	地址总线第10位 — A10
								MISO_2	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P2.3/A11	33	39	35	28	24	26		P2.3	标准I/O口 PORT2[3]
								A11	地址总线第11位 — A11
								MOSI_2	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P2.4/A12	34	40	36	29	25	27		P2.4	标准I/O口 PORT2[4]
								A12	地址总线第12位 — A12
								SS_2	SPI同步串行接口的从机选择信号
P2.5/A13	35	41	37	30	26	28		P2.5	标准I/O口 PORT2[5]
								A13	地址总线第13位 — A13
P2.6/A14	36	42	38	31	27	1		P2.6	标准I/O口 PORT2[6]
								A14	地址总线第14位 — A14
P2.7/A15	37	43	39	32	28	2		P2.7	标准I/O口 PORT2[7]
								A15	地址总线第15位 — A15
P3.0/RxD/ $\overline{\text{INT4}}$ /T2CLKO	18	24	21	17	13	15	11	P3.0	标准I/O口 PORT3[0]
								RxD	串口1数据接收端
								$\overline{\text{INT4}}$	外部中断4,只能下降沿中断, /INT4支持掉电唤醒
								T2CLKO	T2的时钟输出 可通过设置INT_CLKO[2]位 /T2CLKO将该管脚配置为T2CLKO
P3.1/TxD/ T2	19	25	22	18	14	16	12	P3.1	标准I/O口 PORT3[1]
								TxD	串口1数据发送端
								T2	定时器/计数器2的外部输入
P3.2/INT0	20	26	23	19	15	17	13	P3.2	标准I/O口 PORT3[2]
								INT0	外部中断0,既可上升沿中断也可下降沿中断. 如果IT0(TCON.0)被置为1,INT0管脚仅为下降沿中断。如果IT0(TCON.0)被清0，INT0管脚既支持上升沿中断也支持下降沿中断。 INT0支持掉电唤醒。
P3.3/INT1	21	27	24	20	16	18	14	P3.3	标准I/O口 PORT3[3]
								INT1	外部中断1, 既可上升沿中断也可下降沿中断. 如果IT1(TCON.2)被置为1,INT1管脚仅为下降沿中断。如果IT1(TCON.2)被清0，INT1管脚既支持上升沿中断也支持下降沿中断。 INT1支持掉电唤醒。
P3.4/T0/ T1CLKO	22	28	25	21	17	19	15	P3.4	标准I/O口 PORT3[4]
								T0	定时器/计数器0的外部输入
								T1CLKO	定时器/计数器1的时钟输出 可通过设置INT_CLKO[1]位 /T1CLKO将该管脚配置为 T1CLKO，也可对T1脚的外部时钟输入进行分频输出

STC MCU

管脚	管脚编号							说明	
	LQFP44	PLCC44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	TSSOP20		
P3.5/T1/ T0CLKO	23	29	26	22	18	20	16	P3.5	标准I/O口 PORT3[5]
								T1	定时器/计数器1的外部输入
								T0CLKO	定时器/计数器0的时钟输出 可通过设置INT_CLKO[0]位 /T0CLKO将该管脚配置为 T0CLKO，也可对T0脚的外部时钟 输入进行分频输出
P3.6/$\overline{INT2}$ /RxD_2	24	30	27	23	19	21	17	P3.6	标准I/O口 PORT3[6]
								$\overline{INT2}$	外部中断2, 只能下降沿中断 支持掉电唤醒
								RxD_2	串口1数据接收端
P3.7/$\overline{INT3}$ /TxD_2	25	31	28	24	20	22	18	P3.7	标准I/O口 PORT3[7]
								$\overline{INT3}$	外部中断3, 只能下降沿中断 支持掉电唤醒
								TxD_2	串口1数据发送端
P4.0	17	23	-	-	-	-		标准I/O口 PORT4[0]	
P4.1	26	32	29	-	-	-		标准I/O口 PORT4[1]	
P4.2/$\overline{WR}$	27	33	30	-	-	-		P4.2	标准I/O口 PORT4[2]
								$\overline{WR}$	外部数据存储器写脉冲
P4.3	28	34	-	-	-	-		标准I/O口 PORT4[3]	
P4.4/$\overline{RD}$	29	35	31	-	-	-		P4.4	标准I/O口 PORT4[4]
								$\overline{RD}$	外部数据存储器读脉冲
P4.5/ALE	38	44	40	-	-	-		P4.5	标准I/O口 PORT4[5]
								ALE	地址锁存允许
P4.6	39	1	-	-	-	-		标准I/O口 PORT4[6]	
P4.7	6	12	-	-	-	-		标准I/O口 PORT4[7]	
P5.4/RST/ MCLKO/ CMP-	13	19	17	13	9	11	7	P5.4	标准I/O口 PORT5[4]
								RST	复位脚(高电平复位)
								MCLKO	主时钟输出;输出的频率可为 MCLK/1, MCLK/2, MCLK/4. 此系列的主时钟只可以是内部R/C 时钟，MCLK指主时钟频率。
								CMP-	比较器负极输入端
P5.5/CMP+	15	21	19	15	11	13	9	P5.5	标准I/O口 PORT5[5]
								CMP+	比较器正极输入端
Vcc	14	20	18	14	10	12	8	电源正极	
Gnd	16	22	20	16	12	14	10	电源负极，接地	

STC MCU Limited

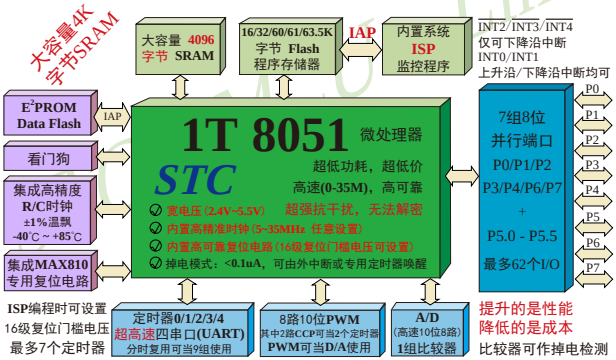
1.8 STC15W4K60S4系列单片机总体介绍(2014年8月开始送样)

1.8.1 STC15W4K60S4系列单片机简介

STC15W4K60S4系列单片机是STC生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/超强抗干扰的新一代8051单片机，采用STC第九代加密技术，无法解密，指令代码完全兼容传统8051,但速度快8-12倍。内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP编程时16级复位门槛电压可选)。8路10位PWM，8路高速10位A/D转换(30万次/秒)，内置4K字节大容量SRAM，4组独立的高速异步串行通信端口(UART1/UART2/UART3/UART4)，1组高速同步串行通信端口SPI，针对多串行口通信/电机控制/强干扰场合。内置比较器，功能更强大。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通8051快8-12倍
2. 工作电压：2.4V - 5.5V
3. 16K/32K/40K/48K/56K/60K/61K/63.5K字节片内Flash程序存储器，擦写次数10万次以上
4. 片内大容量4096字节的SRAM，包括常规的256字节RAM <idata> 和内部扩展的3840字节XRAM <xdata>
5. 大容量片内EEPROM，擦写次数10万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 共8通道10位高速ADC，速度可达30万次/秒，8路PWM还可当8路D/A使用
8. 6通道15位专门的高精度PWM(带死区控制)+2通道CCP(利用它的高速脉冲输出功能可实现11~16位PWM)----可用来再实现8路D/A，或2个16位定时器，或2个外部中断(支持上升沿/下降沿中断)
9. 内部高可靠复位，ISP编程时16级复位门槛电压可选，可彻底省掉外部复位电路

10.内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时内部时钟从5MHz~35MHz可设(5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz)

11.不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号

12.工作频率范围：5MHz ~ 35MHz，相当于普通8051的60MHz～420MHz

13.四组完全独立的高速异步串行通信端口，分时切换可当9组串口使用：

串口1(RxD/P3.0, TxD/P3.1)可以切换到(RxD_2/P3.6, TxD_2/P3.7)，
还可以切换到(RxD_3/P1.6, TxD_3/P1.7)；
串口2(RxD2/P1.0, TxD2/P1.1)可以切换到(RxD2_2/P4.6, TxD2_2/P4.7)
串口3(RxD3/P0.0, TxD3/P0.1)可以切换到(RxD3_2/P5.0, TxD3_2/P5.1)
串口4(RxD4/P0.2, TxD4/P0.3)可以切换到(RxD4_2/P5.2, TxD4_2/P5.3)

14. 一组高速同步串行通信端口SPI.

15.支持程序加密后传输，防拦截

16.支持RS485下载

17. 低功耗设计：低速模式，空闲模式，掉电模式/停机模式.

18. 可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。

19. 可将掉电模式/停机模式唤醒的资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚CCP0/CCP1；外部管脚RxD/RxD2/RxD3/RxD4(下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的串行口中断已经被允许)；外部管脚T0/T1/T2/T3/T4(下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；内部低功耗掉电唤醒专用定时器。

20. 共7个定时器，5个16位可重装载定时器/计数器(T0/T1/T2/T3/T4，其中T0/T1兼容普通8051的定时器/计数器)，并均可独立实现对外可编程时钟输出(5通道)，另外管脚MCLKO可将内部主时钟对外分频输出(÷1或÷2或÷4)，2路CCP还可再实现2个定时器

21. 定时器/计数器2，也可实现1个16位重装载定时器/计数器,定时器/计数器2也可产生时钟输出T2CLKO

22. 新增可16位重装载定时器T3/T4，也可产生可编程时钟输出T3CLKO/T4CLKO

23. 可编程时钟输出功能(对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出)：
由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz.；
而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz.

STC MCU Limited

① T0在P3.5/T0CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出);

② T1在P3.4/T1CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出);

③ T2在P3.0/T2CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出);

④ T3在P0.4/T3CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T3/P0.5的时钟输入进行可编程时钟分频输出);

⑤ T4在P0.6/T4CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T4/P0.7的时钟输入进行可编程时钟分频输出);

以上5个定时器/计数器均可1~65536级分频输出。

⑥ 主时钟在P5.4/MCLKO或P1.6/XTAL2/MCLKO_2对外输出时钟，并可如下分频MCLK/1, MCLK/2, MCLK/4.

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。MCLK是指主时钟频率，MCLKO是指主时钟输出。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列等)均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

24. 比较器，可当1路ADC使用，可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出（可设置极性），也支持外部管脚CMP+与内部参考电压进行比较

25. 硬件看门狗(WDT)

26. 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令

27. 通用I/O口(62/46/42/38/30/26个)，复位后为：准双向口/弱上拉（普通8051传统I/O口）
可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
每个I/O口驱动能力均可达到20mA，但40-pin及40-pin以上单片机的整个芯片电流最大不要超过120mA，16-pin及以上/32-pin及以下单片机的整个芯片电流最大不要超过90mA.
如果I/O口不够用，可外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口

28. 封装：LQFP64L(16mm x 16mm), LQFP64S(12mm x 12mm), LQFP48(9mm x 9mm), LQFP44(12mm x 12mm), LQFP32(9mm x 9mm), SOP28, SKDIP28, PDIP40.

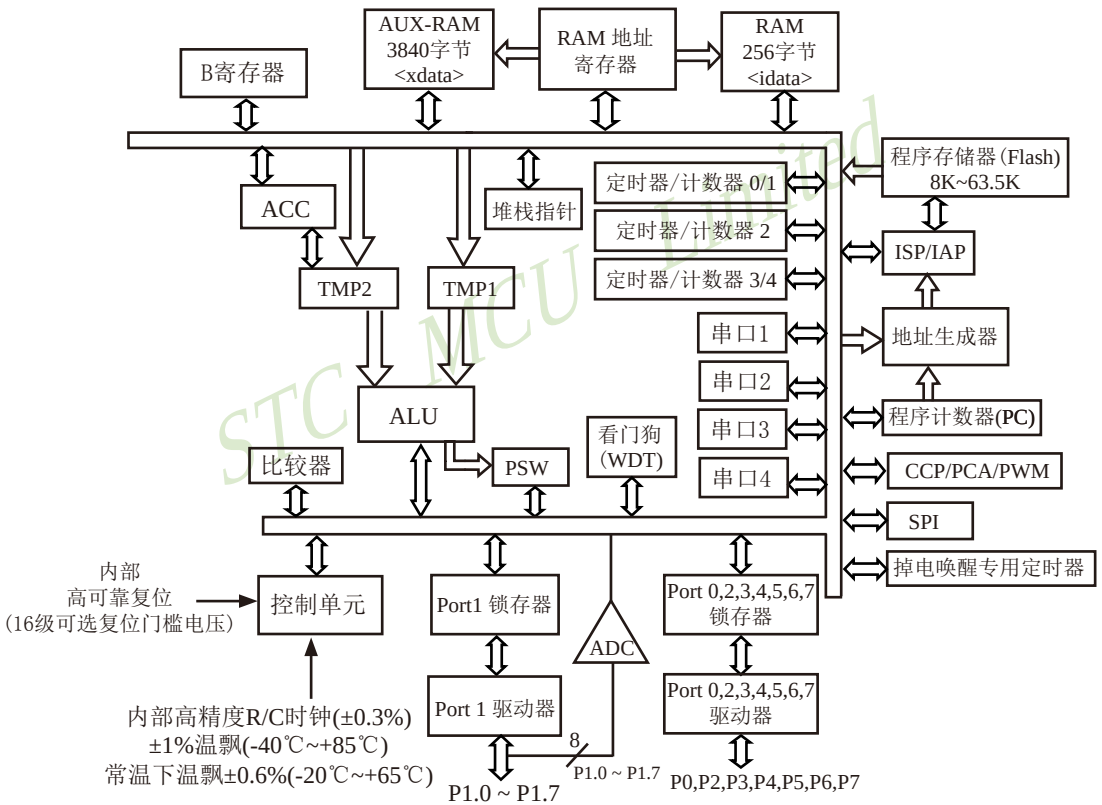
29. 全部175℃八小时高温烘烤，高品质制造保证

30. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

STC MCU Limited

1.8.2 STC15W4K60S4系列单片机的内部结构图

STC15W4K60S4系列单片机的内部结构框图如下图所示。STC15W4K60S4系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、掉电唤醒专用定时器、I/O口、高速A/D转换、比较器、看门狗、UART高速异步串行通信口1、串行口2、串行口3、串行口4、CCP/PWM/PCA、高速同步串行通信端口SPI，片内高精度R/C时钟及高可靠复位等模块。STC15W4K60S4系列单片机几乎包含了数据采集和控制中所需要的所有单元模块，可称得上是一个真正的片上系统(SysTem Chip或SysTem on Chip, 简写为STC, 这是宏晶科技STC名称的由来)。



STC15W4K60S4系列内部结构框图

1.8.3 STC15W4K60S4系列单片机管脚图

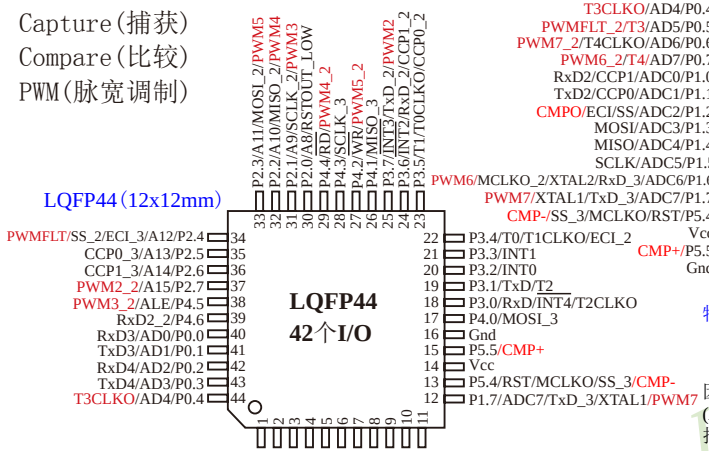
所有封装形式均满足欧盟RoHS要求，

CCP：是英文单词的缩写

Capture(捕获)

Compare(比较)

PWM(脉宽调制)



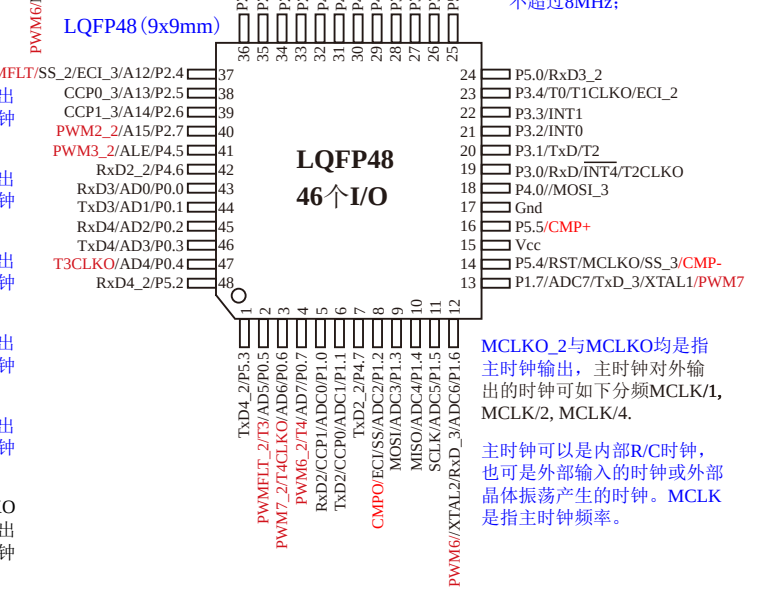
特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...



T0CLKO是指定时器/计数器0的可编程时钟输出(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；

T1CLKO是指定时器/计数器1的可编程时钟输出(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出)；

T2CLKO是指定时器/计数器2的可编程时钟输出(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；

T3CLKO是指定时器/计数器3的可编程时钟输出(对内部系统时钟或对外部管脚T3/P0.5的时钟输入进行可编程时钟分频输出)；

T4CLKO是指定时器/计数器4的可编程时钟输出(对内部系统时钟或对外部管脚T4/P0.7的时钟输入进行可编程时钟分频输出).

T0CLKO/T1CLKO/T2CLKO/T3CLKO/T4CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T1/T2/T3/T4的时钟输入进行时钟分频输出，作分频器使用。

MCLKO_2与MCLKO均是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4.

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。MCLK是指主时钟频率。

STC MCU Limited

T0CLKO是指定时器/计数器0的可编程时钟输出(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出);
T1CLKO是指定时器/计数器1的可编程时钟输出(对内部系统时钟或对外部管脚T1/P3.5的时钟输入进行可编程时钟分频输出);
T2CLKO是指定时器/计数器2的可编程时钟输出(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出);
T3CLKO是指定时器/计数器3的可编程时钟输出
(对内部系统时钟或对外部管脚T3/P0.5的时钟输入进行可编程时钟分频输出);
T4CLKO是指定时器/计数器4的可编程时钟输出(对内部系统时钟或对外部管脚T4/P0.7的时钟输入进行可编程时钟分频输出).

T0CLKO/T1CLKO/T2CLKO/T3CLKO/T4CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T1/T2/T3/T4的时钟输入进行时钟分频输出，作分频器使用。

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz;

对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz;

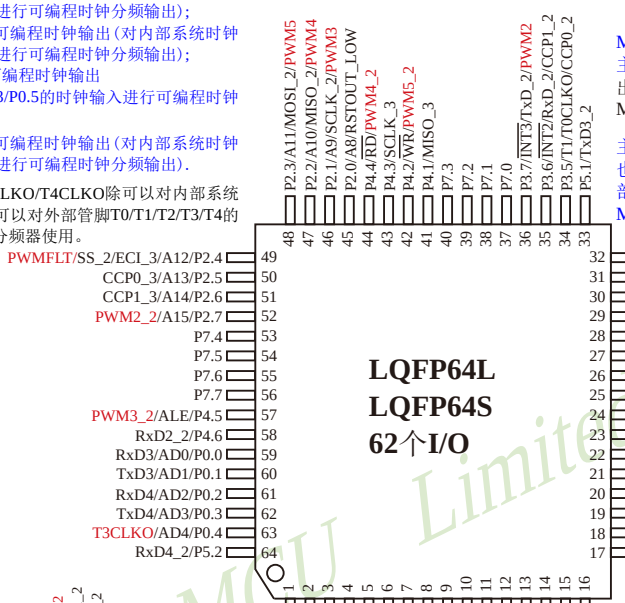
A/D转换通道在P1口，管脚图中P1.x/ADCx是指P1.x管脚可作为A/D转换通道使用。

所有封装形式均满足欧盟RoHS要求，

CCP：是Capture(捕获),Compare(比较),PWM(脉宽调制)的缩写

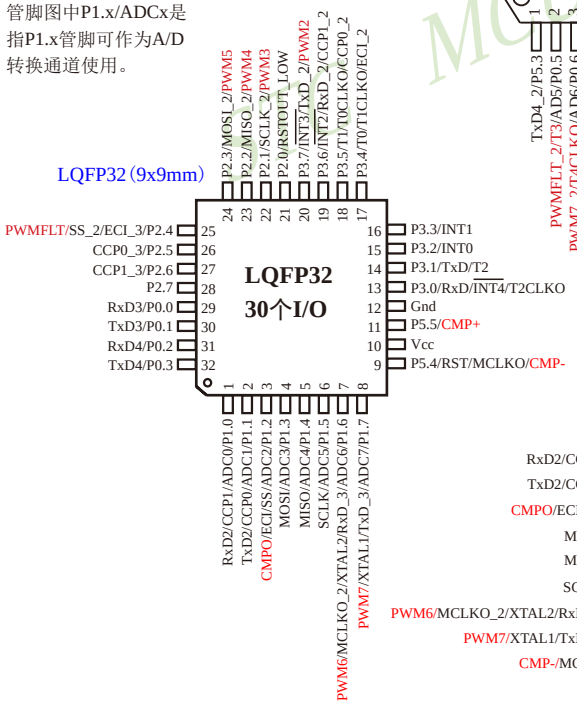
MCLKO_2与MCLKO均是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4.

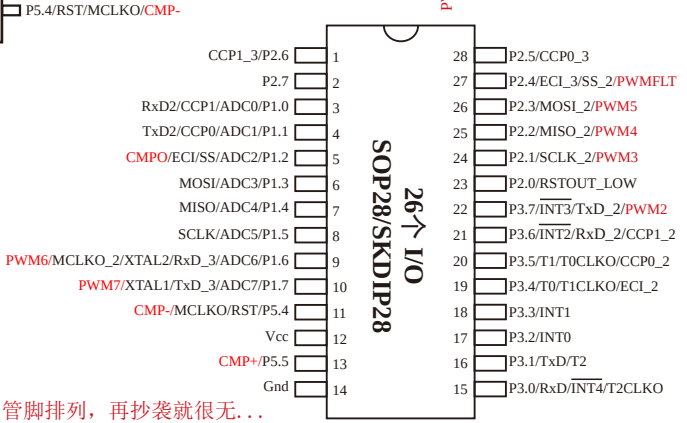
主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。
MCLK是指主时钟频率。



LQFP64L(16x16mm)
LQFP64S(12x12mm)

LQFP32(9x9mm)





中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

MCU Limited

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000

串口1/S1可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD,P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2,P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1] 串口1在P1口时要使用内部时钟
1	1	无效

串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1]上。

建议用户在程序中将[S1_S1, S1_S0]的值设置为[0, 1]或[1, 0]，进而将串口1放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1]上

CCP可在3个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP可在P1/P2/P3之间来回切换
0	0	CCP在[P1.2/ECI,P1.1/CCP0,P1.0/CCP1]
0	1	CCP在[P3.4/ECI_2,P3.5/CCP0_2,P3.6/CCP1_2]
1	0	CCP在[P2.4/ECI_3,P2.5/CCP0_3,P2.6/CCP1_3]
1	1	无效

PWM2/PWM3/PWM4/PWM5/PWMFLT可在2个地方切换，由 PWM2345_S 控制位来选择	
PWM2345_S	切换PWM2/PWM3/PWM4/PWM5/PWMFLT管脚
0	PWM2/PWM3/PWM4/PWM5/PWMFLT在[P3.7/PWM2, P2.1/PWM3, P2.2/PWM4, P2.3/PWM5, P2.4/PWMFLT]
1	PWM2/PWM3/PWM4/PWM5/PWMFLT在[P2.7/PWM2_2, P4.5/PWM3_2, P4.4/PWM4_2, P4.2/PWM5_2, P0.5/PWMFLT_2]

PWM6/PWM7可在2个地方切换，由 PWM67_S 控制位来选择	
PWM67_S	切换PWM6/PWM7管脚
0	PWM6/PWM7在[P1.6/PWM6,P1.7/PWM7]
1	PWM6/PWM7在[P0.7/PWM6_2,P0.6/PWM7_2]

STC MCU Limited

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000

SPI可在3个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI可在P1/P2/P4之间来回切换
0	0	SPI在[P1.2/SS,P1.3/MOSI,P1.4/MISO,P1.5/SCLK]
0	1	SPI在[P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI在[P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	无效

DPS：DPTR registers select bit. DPTR 寄存器选择位

0: DPTR0 is selected　　DPTR0被选择

1: DPTR1 is selected　　DPTR1被选择

串口2/S2可在2个地方切换，由 S2_S 控制位来选择	
S2_S	S2可在P1/P4之间来回切换
0	串口2/S2在[P1.0/RxD2,P1.1/TxD2]
1	串口2/S2在[P4.6/RxD2_2,P4.7/TxD2_2]

串口3/S3可在2个地方切换，由 S3_S 控制位来选择	
S3_S	S3可在P0/P5之间来回切换
0	串口3/S3在[P0.0/RxD3,P0.1/TxD3]
1	串口3/S3在[P5.0/RxD3_2,P5.1/TxD3_2]

串口4/S4可在2个地方切换，由 S4_S 控制位来选择	
S4_S	S4可在P0/P5之间来回切换
0	串口4/S4在[P0.2/RxD4,P0.3/TxD4]
1	串口4/S4在[P5.2/RxD4_2,P5.3/TxD4_2]

STC MCU Limited

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， 输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， 输出时钟频率 = MCLK / 4

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK是指主时钟频率。
STC15W4K60S4系列单片机在MCLKO/P5.4口或MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列)均在MCLKO/P5.4口对外输出主时钟,且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15W4K60S4系列单片机通过CLK_DIV.3/MCLKO_2位来选择是在MCLKO/P5.4口对外输出主时钟，还是在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位
　0：在MCLKO/P5.4口对外输出主时钟；
　1：在MCLKO_2/XTAL2/P1.6口对外输出主时钟；
主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

ADRJ：ADC转换结果调整
　0：ADC_RES[7:0]存放高8位ADC结果，ADC_RESL[1:0]存放低2位ADC结果
　1：ADC_RES[1:0]存放高2位ADC结果，ADC_RESL[7:0]存放低8位ADC结果

Tx_Rx：串口1的中继广播方式设置
　0：串口1为正常工作方式
　1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。
串口1的RxD管脚和TxD管脚可以在3组不同管脚之间进行切换：［RxD/P3.0, TxD/P3.1］;
［RxD_2/P3.6, TxD_2/P3.7］;
［RxD_3/P1.6, TxD_3/P1.7］.

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D转换的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

STC MCU Limited

1.8.4 STC15W4K60S4系列单片机选型价格一览表

型号	工作电压（V）	Flash程序存储器(byte)	大容量SRAM字节	串行口并可掉电唤醒	SPI	普通定时器计数器T0-T4外部管脚也能掉电唤醒	8路 PWM		掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(2路CCP可当3路D/A使用)	比较器(可当1路A/D使用,可作外部掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	所有封装 LQFP64/LQFP48/LQFP44/ PDIP40 LQFP32/SOP28/SKDIP28 部分封装 价格(RMB ¥)			
							6路15位专门的PWM(带死区控制)	2路CCP 10位PWM可当外部中断并可掉电唤醒														PDIP40	LQFP44	LQFP48	LQFP64
STC15W4K60S4系列单片机选型价格一览表，2014年8月开始供货 特别提醒：8路PWM可当8路D/A使用,2路CCP可当2个定时器或2个外部中断使用																									
STC15W4K16S4	2.4-5.5	16K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	45K	有	有	16级	有	是	有	是	¥5.5	¥5.0	¥5.2	¥6.0
STC15W4K32S4	2.4-5.5	32K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	29K	有	有	16级	有	是	有	是	¥6.0	¥5.5	¥5.7	¥6.5
STC15W4K40S4	2.4-5.5	40K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	21K	有	有	16级	有	是	有	是	¥6.5	¥6.0	¥6.2	¥7.0
STC15W4K48S4	2.4-5.5	48K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	13K	有	有	16级	有	是	有	是	¥6.5	¥6.0	¥6.2	¥7.0
STC15W4K56S4	2.4-5.5	56K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	5K	有	有	16级	有	是	有	是	¥6.5	¥6.0	¥6.2	¥7.0
STC15W4K60S4	2.4-5.5	60K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	1K	有	有	16级	有	是	有	是	¥7.0	¥6.5	¥6.7	¥7.5
IAP15W4K61S4 本身就是仿真器	2.4-5.5	61K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	IAP	有	有	16级	有	是	有	是	¥7.0	¥6.5	¥6.7	¥7.5
																						用户可将用户程序区的程序Flash当EEPROM使用			
IRC15W4K63S4 使用外部晶振或内部24MHz时钟	2.4-5.5	63.5K	4K	4	有	5	6-ch	2-ch	有	有	10位	√	2	IAP	有	有	固定	有	是	无	否	¥7.0	¥6.5	¥6.7	¥7.5
																						用户可将用户程序区的程序Flash当EEPROM使用			

提供客制化IC服务

如果用户要用40-pin及以上的单片机，建议选用LQFP44的封装，但PDIP40封装仍正常供货；如果用户要用32-pin单片机，建议用户选用LQFP32封装；如果用户要用28-pin单片机，建议用户选用SOP28封装。

以上单价为10K起订
量小每片需加0.1元
以上价格运费由客户承担,零售10片起
以上价格仅供参考，如对价格不满，可来电要求降价

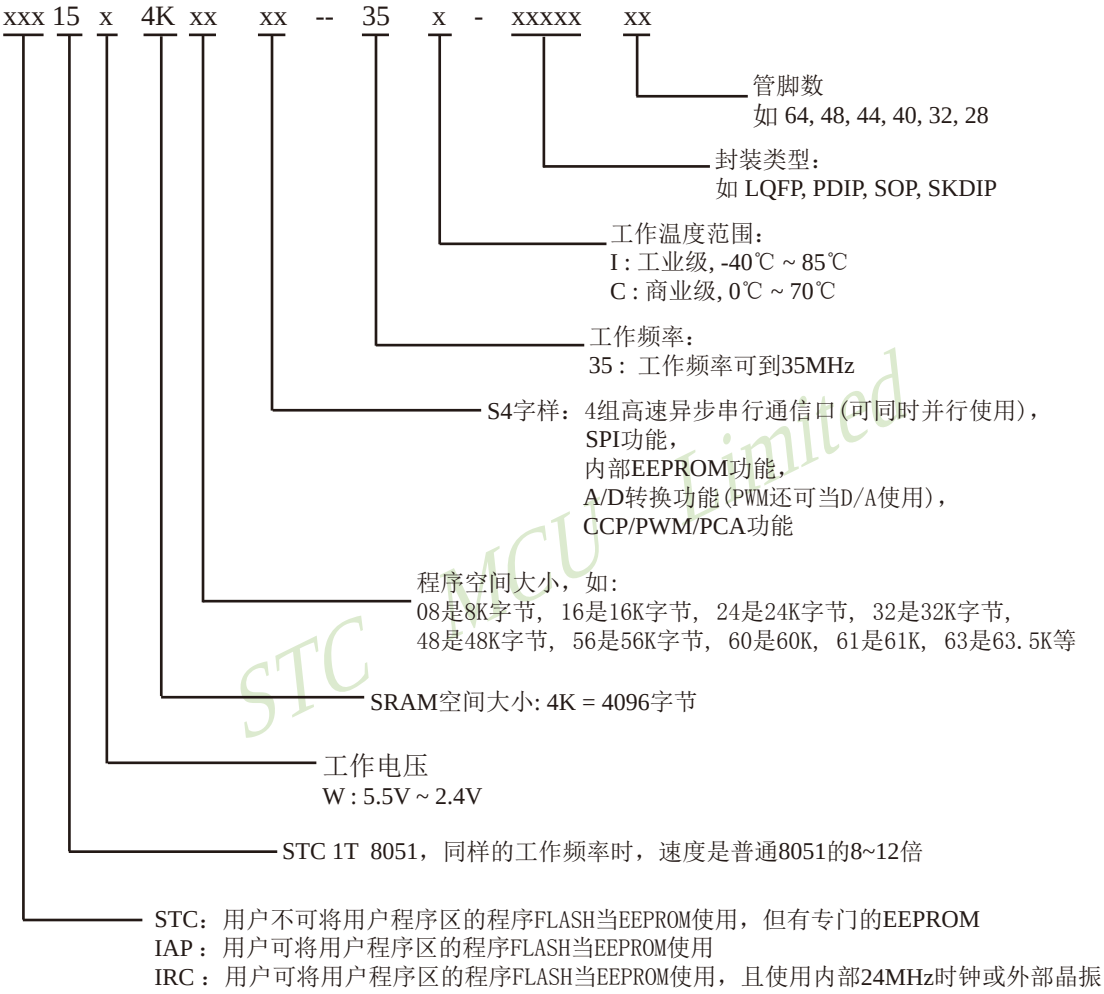
程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

因为程序区的最后7个字节单元被强制性的放入全球唯一ID号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少7个字节。

上表中IRC15W4K63S4型号的单片机固定使用内部24MHz时钟，且其内部复位门槛电压固定，同时不支持"程序加密后传输"功能，其P5.4不可当复位管脚RST使用，且P3.2/P3.3与下载无关。

总结：STC15W4K60S4系列单片机（含IRC15W4K63S4）有：5个16位可重装载普通定时器/计数器，这5个普通定时器/计数器分别是定时器/计数器0、定时器/计数器1、定时器/计数器2、定时器/计数器3和定时器/计数器4；8路10位PWM(可再实现8个D/A转换器或2个定时器或2个外部中断)；掉电唤醒专用定时器；5个外部中断INT0/INT1/INT2/INT3/INT4；4组高速异步串行通信端口(可同时使用)；1组高速同步串行通信端口SPI；8路高速10位A/D转换器；1个比较器；2个数据指针DPTR；外部数据总线等功能。

1.8.5 STC15W4K60S4系列单片机命名规则



命名举例：

（1）STC15W4K32S4 - 35I - SOP28 表示：

用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为4K(4096)字节，程序空间大小为32K，有四组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为28。

（2）STC15W4K40S4 - 35I - SKDIP28 表示：

用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为4K(4096)字节，程序空间大小为40K，有四组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SKDIP封装，管脚数为28。

（3）STC15W4K48S4 - 35I - LQFP32 表示：

用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为4K(4096)字节，程序空间大小为48K，有四组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为32。

（4）STC15W4K60S4 - 35I - LQFP44 表示：

用户不可以将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为4K(4096)字节，程序空间大小为60K，有四组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为LQFP贴片封装，管脚数为44。

（5）IAP15W4K61S4 - 35I - PDIP40 表示：

用户可以将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~2.4V，SRAM空间大小为4K(4096)字节，程序空间大小为61K，有四组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到35MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为PDIP贴片封装，管脚数为40。

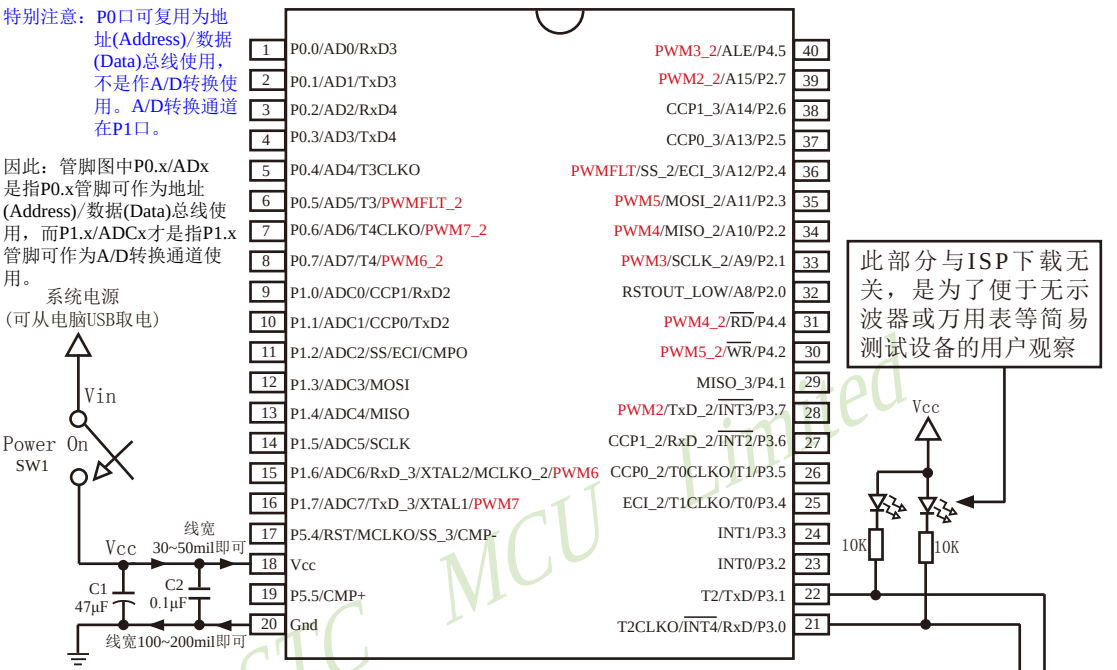
STC MCU Limited

1.8.6 STC15W4K60S4系列单片机在系统可编程(ISP)典型应用线路图

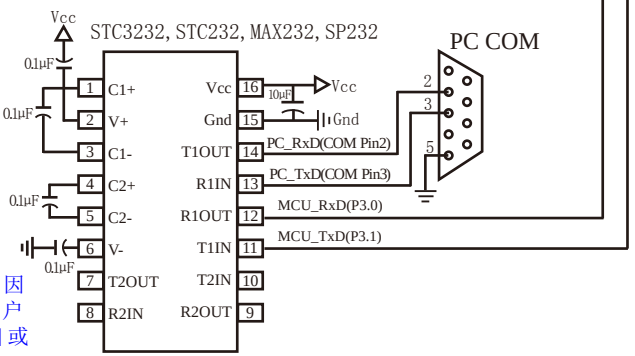
1.8.6.1 利用RS-232转换器的ISP下载编程典型应用线路图

特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。



STC 单片机在线编程线路， ***STC*** **RS-232** 转换器



烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

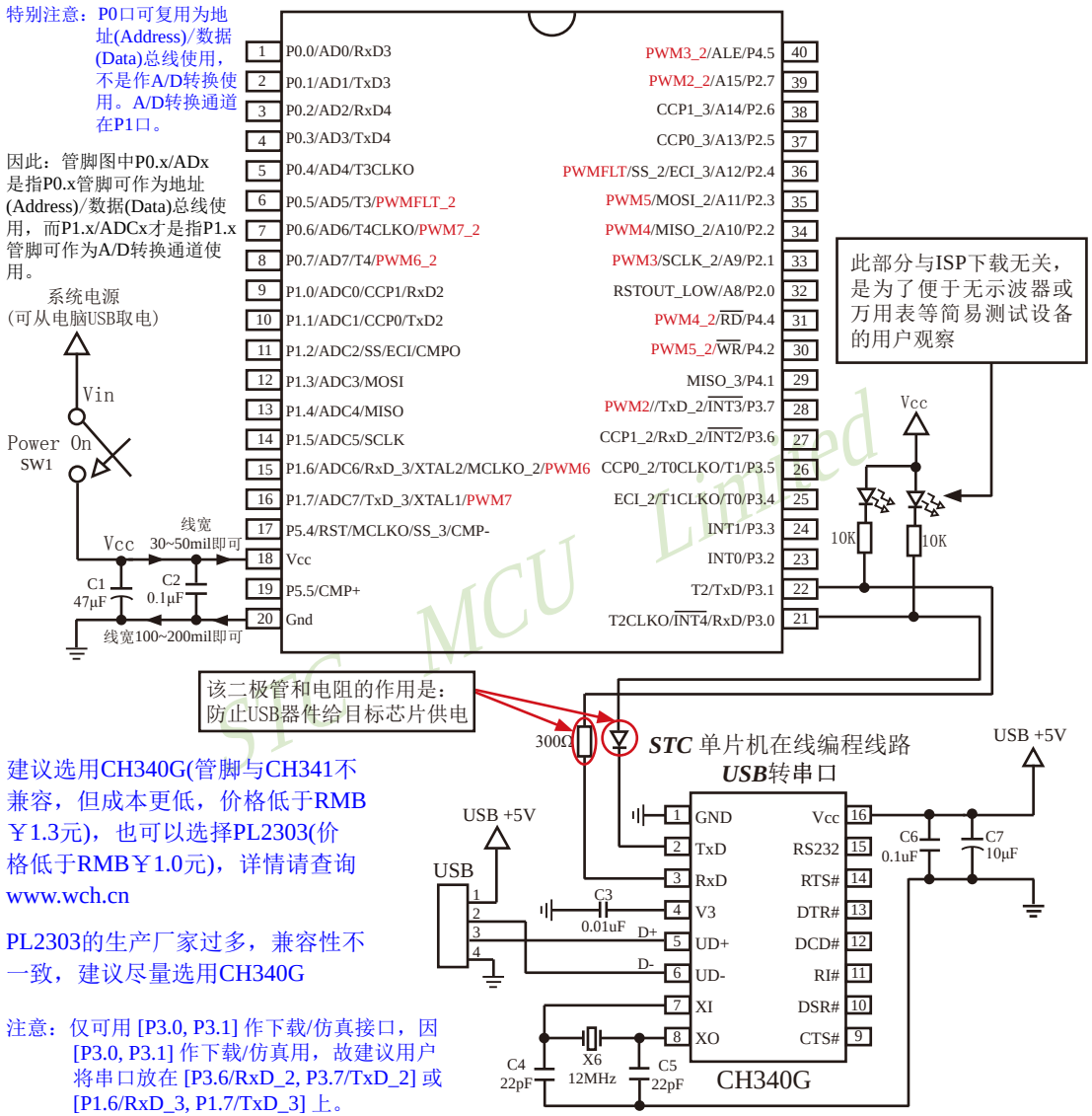
内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.8.6.2 利用USB转串口的ISP下载编程典型应用线路图



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路
P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.8.7 STC15W4K60S4系列单片机的管脚说明

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P0.0/AD0/ RxD3	59	43	40	1	1	29	-	P0.0	标准I/O口 PORT0[0]
								AD0	地址/数据总线
								RxD3	串口3数据接收端
P0.1/AD1/ TxD3	60	44	41	2	2	30	-	P0.1	标准I/O口 PORT0[1]
								AD1	地址/数据总线
								TxD3	串口3数据发送端
P0.2/AD2/ RxD4	61	45	42	3	3	31	-	P0.2	标准I/O口 PORT0[2]
								AD2	地址/数据总线
								RxD4	串口4数据接收端
P0.3/AD3/ TxD4	62	46	43	4	4	32	-	P0.3	标准I/O口 PORT0[3]
								AD3	地址/数据总线
								TxD4	串口4数据发送端
P0.4/AD4/ T3CLKO	63	47	44	5	-	-	-	P0.4	标准I/O口 PORT0[4]
								AD4	地址/数据总线
								T3CLKO	定时器/计数器3的时钟输出可通过设置T4T3M[0]位/T3CLKO将该管脚配置为T3CLKO
P0.5/ AD5/T3/ PWMFLT_2	2	2	1	6	-	-	-	P0.5	标准I/O口 PORT0[5]
								AD5	地址/数据总线
								T3	定时器/计数器3的外部输入
								PWMFLT_2	PWM异常停机控制管脚
P0.6/AD6/ T4CLKO/ PWM7_2	3	3	2	7	-	-	-	P0.6	标准I/O口 PORT0[6]
								AD6	地址/数据总线
								T4CLKO	定时器/计数器4的时钟输出可通过设置T4T3M[4]位/T4CLKO将该管脚配置为T4CLKO
								PWM7_2	脉宽调制输出通道-7
P0.7/ AD7/T4/ PWM6_2	4	4	3	8	-	-	-	P0.7	标准I/O口 PORT0[7]
								AD7	地址/数据总线
								T4	定时器/计数器4的外部输入
								PWM6_2	脉宽调制输出通道-6
P1.0/ADC0/ CCP1/RxD2	9	5	4	9	5	1	3	P1.0	标准I/O口 PORT1[0]
								ADC0	ADC 输入通道-0
								CCP1	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-1
								RxD2	串口2数据接收端

STC MCU Limited

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P1.1/ADC1/ CCP0/TxD2	10	6	5	10	6	2	4	P1.1	标准I/O口 PORT1[1]
								ADC1	ADC 输入通道-1
								CCP0	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-0
								TxD2	串口2数据发送端
P1.2/ADC2/ SS/ECI/ CMPO	12	8	7	11	7	3	5	P1.2	标准I/O口 PORT1[2]
								ADC2	ADC 输入通道-2
								SS	SPI同步串行接口的从机选择信号
								ECI	CCP / PCA计数器的外部脉冲输入脚
								CMPO	比较器的比较结果输出管脚
P1.3/ADC3/ MOSI	13	9	8	12	8	4	6	P1.3	标准I/O口 PORT1[3]
								ADC3	ADC 输入通道-3
								MOSI	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P1.4/ADC4/ MISO	14	10	9	13	9	5	7	P1.4	标准I/O口 PORT1[4]
								ADC4	ADC 输入通道-4
								MISO	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P1.5/ADC5/ SCLK	15	11	10	14	10	6	8	P1.5	标准I/O口 PORT1[5]
								ADC5	ADC 输入通道-5
								SCLK	SPI同步串行接口的时钟信号
P1.6/ADC6/ RxD_3/ XTAL2/ MCLKO_2/ PWM6	16	12	11	15	11	7	9	P1.6	标准I/O口 PORT1[6]
								ADC6	ADC 输入通道-6
								RxD_3	串口1数据接收端
								MCLKO_2	主时钟输出;输出的频率可为MCLK/1, MCLK/2, MCLK/4. 主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK指主时钟频率。
								XTAL2	内部时钟电路反相放大器的输出端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚可浮空，此时XTAL2实际将XTAL1输入的时钟进行输出。
								PWM6	脉宽调制输出通道-6

STC MCU Limited

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P1.7/ADC7/ TxD_3/ XTAL1/ PWM7	17	13	12	16	12	8	10	P1.7	标准I/O口 PORT1[7]
								ADC7	ADC 输入通道-7
								TxD_3	串口1数据发送端
								XTAL1	内部时钟电路反相放大器输入端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚是外部时钟源的输入端。
								PWM7	脉宽调制输出通道-7
P2.0/A8/ RSTOUT_LOW	45	33	30	32	25	21	23	P2.0	标准I/O口 PORT2[0]
								A8	地址总线第8位 — A8
								RSTOUT_LOW	上电后,输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P2.1/A9/ SCLK_2/ PWM3	46	34	31	33	26	22	24	P2.1	标准I/O口 PORT2[1]
								A9	地址总线第9位 — A9
								SCLK_2	SPI同步串行接口的时钟信号
								PWM3	脉宽调制输出通道-3
P2.2/A10/ MISO_2/ PWM4	47	35	32	34	27	23	25	P2.2	标准I/O口 PORT2[2]
								A10	地址总线第10位 — A10
								MISO_2	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
								PWM4	脉宽调制输出通道-4
P2.3/A11/ MOSI_2/ PWM5	48	36	33	35	28	24	26	P2.3	标准I/O口 PORT2[3]
								A11	地址总线第11位 — A11
								MOSI_2	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
								PWM5	脉宽调制输出通道-5
P2.4/A12/ ECI_3/SS_2/ PWMFLT	49	37	34	36	29	25	27	P2.4	标准I/O口 PORT2[4]
								A12	地址总线第12位 — A12
								ECI_3	CCP / PCA计数器的外部脉冲输入脚
								SS_2	SPI同步串行接口的从机选择信号
								PWMFLT	PWM异常停机控制管脚
P2.5/A13/ CCP0_3	50	38	35	37	30	26	28	P2.5	标准I/O口 PORT2[5]
								A13	地址总线第13位 — A13
								CCP0_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0

STC MCU Limited

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P2.6/A14/ CCP1_3	51	39	36	38	31	27	1	P2.6	标准I/O口 PORT2[6]
								A14	地址总线第14位 — A14
								CCP1_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
P2.7/A15/ PWM2_2	52	40	37	39	32	28	2	P2.7	标准I/O口 PORT2[7]
								A15	地址总线第15位 — A15
								PWM2_2	脉宽调制输出通道-2
P3.0/RxD/$\overline{\text{INT4}}$ /T2CLKO	27	19	18	21	17	13	15	P3.0	标准I/O口 PORT3[0]
								RxD	串口1数据接收端
								$\overline{\text{INT4}}$	外部中断4, 只能下降沿中断, $\overline{\text{INT4}}$支持掉电唤醒
								T2CLKO	T2的时钟输出 可通过设置INT_CLKO[2]位/T2CLKO将该管脚配置为T2CLKO
P3.1/TxD/T2	28	20	19	22	18	14	16	P3.1	标准I/O口 PORT3[1]
								TxD	串口1数据发送端
								T2	定时器/计数器2的外部输入
P3.2/INT0	29	21	20	23	19	15	17	P3.2	标准I/O口 PORT3[2]
								INT0	外部中断0，既可上升沿中断也可下降沿中断. 如果IT0(TCON.0)被置为1, INT0管脚仅为下降沿中断。如果IT0(TCON.0)被清0，INT0管脚既支持上升沿中断也支持下降沿中断。 INT0支持掉电唤醒。
P3.3/INT1	30	22	21	24	20	16	18	P3.3	标准I/O口 PORT3[3]
								INT1	外部中断1，既可上升沿中断也可下降沿中断. 如果IT1(TCON.2)被置为1, INT1管脚仅为下降沿中断。如果IT1(TCON.2)被清0，INT1管脚既支持上升沿中断也支持下降沿中断。 INT1支持掉电唤醒。
P3.4/T0/ T1CLKO/ ECI_2	31	23	22	25	21	17	19	P3.4	标准I/O口 PORT3[4]
								T0	定时器/计数器0的外部输入
								T1CLKO	定时器/计数器1的时钟输出 可通过设置INT_CLKO[1]位/T1CLKO将该管脚配置为T1CLKO，也可对T1脚的外部时钟输入进行分频输出
								ECI_2	CCP/PCA计数器的外部脉冲输入脚

STC MCU Limited

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P3.5/T1/T0CLKO/ CCP0_2	34	26	23	26	22	18	20	P3.5	标准I/O口 PORT3[5]
								T1	定时器/计数器1的外部输入
								T0CLKO	定时器/计数器0的时钟输出可通过设置INT_CLKO[0]位/T0CLKO将该管脚配置为T0CLKO，也可对T0脚的外部时钟输入进行分频输出
								CCP0_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P3.6/$\overline{\text{INT2}}$/ RxD_2/CCP1_2	35	27	24	27	23	19	21	P3.6	标准I/O口 PORT3[6]
								$\overline{\text{INT2}}$	外部中断2, 只能下降沿中断 $\overline{\text{INT2}}$支持掉电唤醒
								RxD_2	串口1数据接收端
								CCP1_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
P3.7/$\overline{\text{INT3}}$/TxD_2/ PWM2	36	28	25	28	24	20	22	P3.7	标准I/O口 PORT3[7]
								$\overline{\text{INT3}}$	外部中断3, 只能下降沿中断 $\overline{\text{INT3}}$支持掉电唤醒
								TxD_2	串口1数据发送端
								PWM2	脉宽调制输出通道-2
P4.0/MOSI_3	22	18	17	-	-	-	-	P4.0	标准I/O口 PORT4[0]
								MISO_3	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P4.1/MISO_3	41	29	26	29	-	-	-	P4.1	标准I/O口 PORT4[1]
								MOSI_3	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P4.2/$\overline{\text{WR}}$ /PWM5_2	42	30	27	30	-	-	-	P4.2	标准I/O口 PORT4[2]
								$\overline{\text{WR}}$	外部数据存储器写脉冲
								PWM5_2	脉宽调制输出通道-5
P4.3/SCLK_3	43	31	28	-	-	-	-	P4.3	标准I/O口 PORT4[3]
								SCLK_3	SPI同步串行接口的时钟信号
P4.4/$\overline{\text{RD}}$/PWM4_2	44	32	29	31	-	-	-	P4.4	标准I/O口 PORT4[4]
								$\overline{\text{RD}}$	外部数据存储器读脉冲
								PWM4_2	脉宽调制输出通道-4

STC MCU Limited

管脚	管脚编号							说明	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P4.5/ALE/ PWM3_2	57	41	38	40	-	-	-	P4.5	标准I/O口 PORT4[5]
								ALE	地址锁存允许
								PWM3_2	脉宽调制输出通道-3
P4.6/RxD2_2	58	42	39	-	-	-	-	P4.6	标准I/O口 PORT4[6]
								RxD2_2	串口2数据接收端
P4.7/TxD2_2	11	7	6	-	-	-	-	P4.7	标准I/O口 PORT4[7]
								TxD2_2	串口2数据发送端
P5.0/RxD3_2	32	24	-	-	-	-	-	P5.0	标准I/O口 PORT5[0]
								RxD3_2	串口3数据接收端
P5.1/TxD3_2	33	25	-	-	-	-	-	P5.1	标准I/O口 PORT5[1]
								TxD3_2	串口3数据发送端
P5.2/RxD4_2	64	48	-	-	-	-	-	P5.2	标准I/O口 PORT5[2]
								RxD4_2	串口4数据接收端
P5.3/TxD4_2	1	1	-	-	-	-	-	P5.3	标准I/O口 PORT5[3]
								TxD4_2	串口4数据发送端
P5.4/RST/ MCLKO/ SS_3/CMP-	18	14	13	17	13	9	11	P5.4	标准I/O口 PORT5[4]
								RST	复位脚(高电平复位)
								MCLKO	主时钟输出;输出的频率可为MCLK/1, MCLK/2, MCLK/4(MCLK为主时钟频率) 主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。
								SS_3	SPI同步串行接口的从机选择信号
								CMP-	比较器负极输入端
P5.5/CMP+	20	16	15	19	15	11	13	P5.5	标准I/O口 PORT5[5]
								CMP+	比较器正极输入端
P6.0	5							标准I/O口 PORT6[0]	
P6.1	6							标准I/O口 PORT6[1]	
P6.2	7							标准I/O口 PORT6[2]	
P6.3	8							标准I/O口 PORT6[3]	
P6.4	23							标准I/O口 PORT6[4]	
P6.5	24							标准I/O口 PORT6[5]	
P6.6	25							标准I/O口 PORT6[6]	
P6.7	26							标准I/O口 PORT6[7]	
P7.0	37							标准I/O口 PORT7[0]	
P7.1	38							标准I/O口 PORT7[1]	

STC MCU Limited

管脚	管脚编号							说明
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	
P7.2	39							标准I/O口 PORT7[2]
P7.3	40							标准I/O口 PORT7[3]
P7.4	53							标准I/O口 PORT7[4]
P7.5	54							标准I/O口 PORT7[5]
P7.6	55							标准I/O口 PORT7[6]
P7.7	56							标准I/O口 PORT7[7]
Vcc	19	15	14	18	14	10	12	电源正极
Gnd	21	17	16	20	16	12	14	电源负极，接地

STC MCU Limited

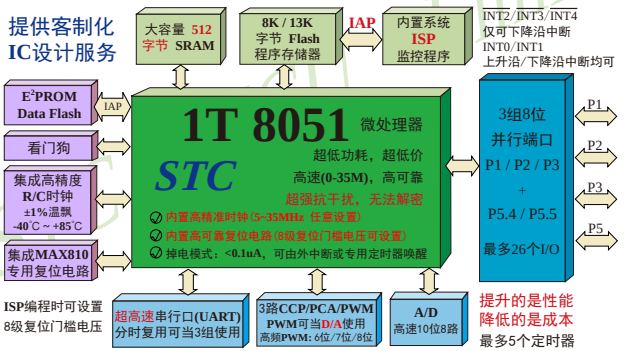
1.9 STC15F408AD系列单片机总体介绍(大批量现货供应中)

1.9.1 STC15F408AD系列单片机简介

STC15F408AD系列单片机是STC生产的单时钟/机器周期(1T)的单片机，是高速/高可靠/低功耗/超强抗干扰的新一代8051单片机，采用STC第八代加密技术，无法解密，指令代码完全兼容传统8051,但速度快8-12倍。内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP编程时8级复位门槛电压可选)。3路CCP/PWM/PCA，8路高速10位A/D转换(30万次/秒)，1组高速异步串行通信口(UART，可在3组管脚之间进行切换，分时复用可作3组串口使用)，1组高速同步串行通信端口SPI，针对串行口通信/电机控制/强干扰场合。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通8051快8-12倍
2. 工作电压：
   STC15F408AD 系列工作电压：5.5V - 4.2V（5V 单片机）
   STC15L408AD 系列工作电压：3.6V - 2.4V（3V 单片机）
3. 8K / 13K字节片内Flash程序存储器，擦写次数10万次以上
4. 片内集成512字节的SRAM，包括常规的256字节RAM <idata> 和内部扩展的256字节XRAM <xdata>
5. 有片内EEPROM功能，擦写次数10万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 共8通道10位高速ADC，速度可达30万次/秒，3路PWM还可当3路D/A使用
8. 共3通道捕获/比较单元(CCP/PWM/PCA)
   -----也可用来再实现3个定时器或3个外部中断(支持上升沿/下降沿中断)或3路D/A

9. 利用CCP/PCA高速脉冲输出功能可实现3路9~16位PWM (每通道占用系统时间小于0.6%)

10.利用定时器T0的时钟输出功能可实现高精度的8~16位PWM (占用系统时间小于0.4%)

11.内部高可靠复位，ISP编程时8级复位门槛电压可选，可彻底省掉外部复位电路

12.内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，ISP编程时内部时钟从5MHz~35MHz可设(5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz)

13.不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号

14.工作频率范围：5MHz ~ 28MHz，相当于普通8051的60MHz～336MHz

15.一组高速异步串行通信端口UART，可在3组管脚之间切换，分时复用可当3组串口使用：
串行口(RxD/P3.0, TxD/P3.1)可以切换到(RxD_2/P3.6, TxD_2/P3.7)，
还可以切换到(RxD_3/P1.6, TxD_3/P1.7)；

16.一组高速同步串行通信端口SPI.

17.支持程序加密后传输，防拦截

18.支持RS485下载

19.低功耗设计：低速模式，空闲模式，掉电模式/停机模式。

20.可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。

21.可将掉电模式/停机模式唤醒的资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿及下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚CCP0/CCP1/CCP2；管脚T0/T2(下降沿，不产生中断，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；内部低功耗掉电唤醒专用定时器。

22. 共5个定时器/计数器，2个16位可重装载并可实现可编程时钟输出的定时器/计数器，分别是定时器/计数器0(即T0)和定时器/计数器2(即T2)，3路CCP/PWM/PCA还可再实现3个定时器。

23. 可编程时钟输出功能(对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出)：
由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz.；
而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz.
① T0在P3.5/T0CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出)；
② T2在P3.0/T2CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；
以上2个定时器/计数器均可1~65536级分频输出。

STC MCU Limited

③ 主时钟在P5.4/MCLKO对外输出时钟，并可如下分频MCLK/1, MCLK/2, MCLK/4.

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。MCLK是指主时钟频率，MCLKO是指主时钟输出。

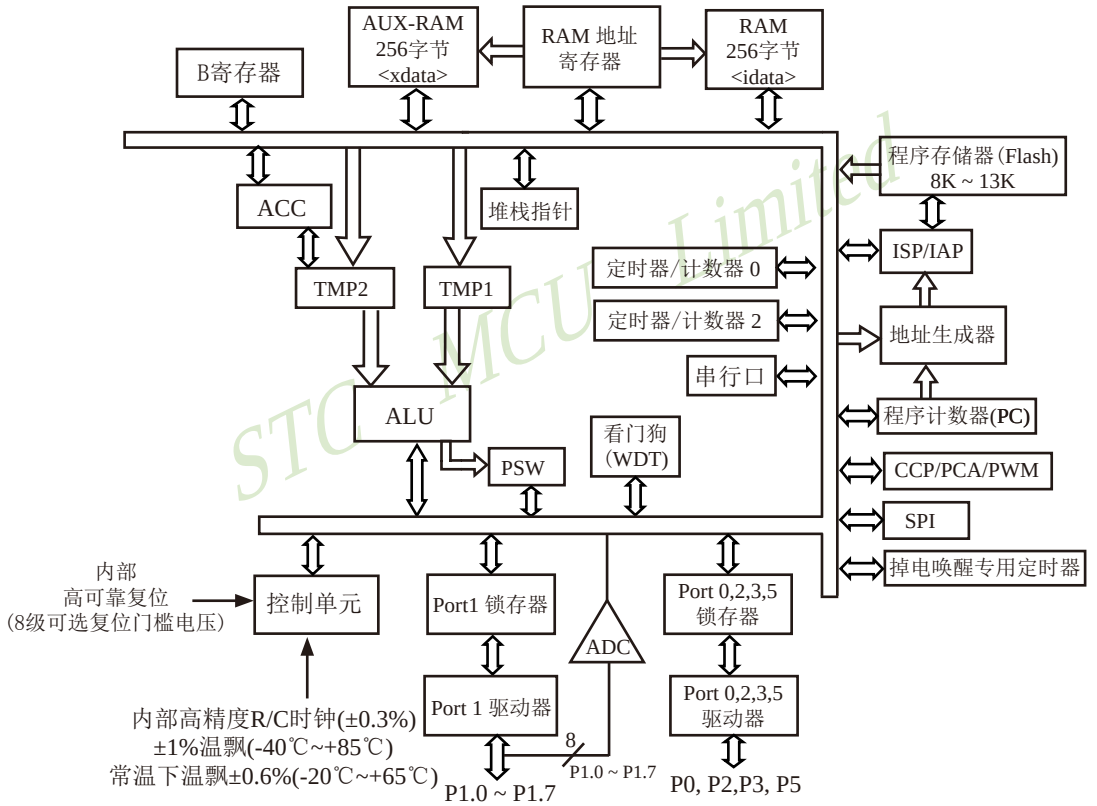
STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/P1.6口对外输出主时钟。

24. 硬件看门狗(WDT)
25. 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
26. 通用I/O口（30/26个），复位后为：准双向口/弱上拉（普通8051传统I/O口）
    可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
    每个I/O口驱动能力均可达到20mA，但整个芯片电流最大不要超过90mA.
    如果I/O口不够用，可外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口
27. 封装：LQFP32(9mm x 9mm), SOP28, SKDIP28 (★此系列的28pin单片机建议用STC15W408AS系列的相应封装的单片机取代).
28. 全部175℃八小时高温烘烤，高品质制造保证
29. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

STC MCU Limited

1.9.2 STC15F408AD系列单片机的内部结构图

STC15F408AD系列单片机的内部结构框图如下图所示。STC15F408AD系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、掉电唤醒专用定时器、I/O口、高速A/D转换(30万次/秒)、看门狗、高速异步串行通信端口UART、CCP/PWM/PCA、高速同步串行端口SPI，片内高精度R/C时钟及高可靠复位等模块。STC15F408AD系列单片机几乎包含了数据采集和控制中所需要的所有单元模块，可称得上是一个片上系统(SysTem Chip或SysTem on Chip,简写为STC,这是宏晶科技STC名称的由来)。



STC15F408AD系列内部结构框图

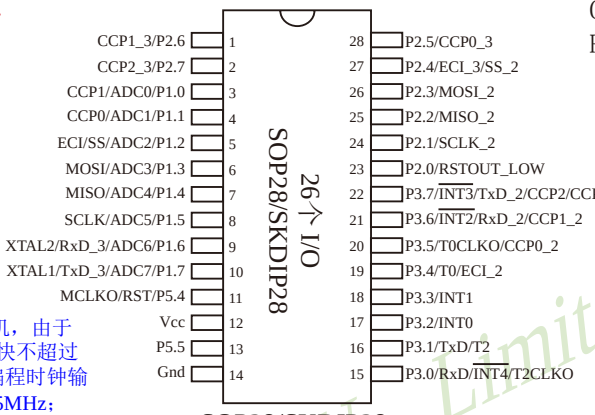
1.9.3 STC15F408AD系列单片机管脚图

所有封装形式均满足欧盟RoHS要求，强烈推荐选择LQFP32贴片封装。

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

CCP：是英文单词的缩写
Capture(捕获),
Compare(比较),
PWM(脉宽调制)

此系列的28pin单片机建议用STC15W408AS系列的相应封装的单片机取代



MCLKO是指主时钟输出，主时钟对外输出的时钟可如下分频MCLK/1, MCLK/2, MCLK/4.

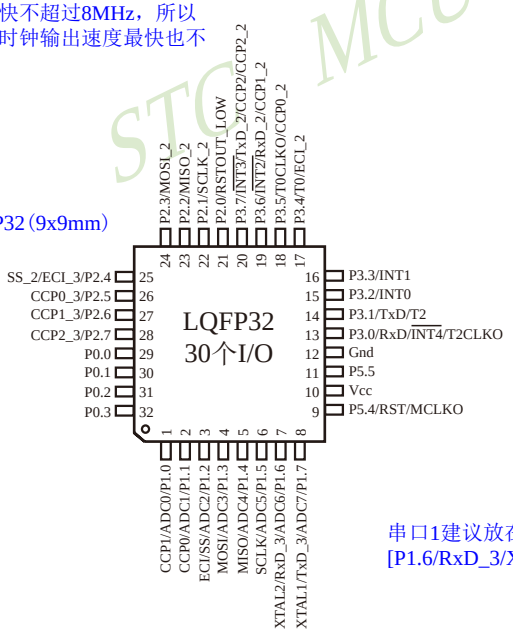
主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。MCLK是指主时钟频率。

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；

对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz；

A/D转换通道在P1口。管脚图中P1.x/ADCx是指P1.x管脚可作为A/D转换通道使用。

LQFP32（9x9mm）



T0CLKO是指定时器/计数器0的可编程时钟输出（对内部系统时钟或对外部管脚T0/P3.4的时钟输入进行可编程时钟分频输出）；

T2CLKO是指定时器/计数器2的可编程时钟输出（对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出）；

T0CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T2的时钟输入进行时钟分频输出，作分频器使用。

串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1]上。

STC MCU Limited

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,x00x
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000,x000

串口1/S1可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD,P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2,P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1] 串口1在P1口时要使用内部时钟
1	1	无效

串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1]上。

建议用户在程序中将[S1_S1, S1_S0]的值设置为[0, 1]或[1, 0]，进而将串口1放在[P3.6/RxD_2, P3.7/TxD_2]或[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1]上

CCP可在3个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP可在P1/P2/P3之间来回切换
0	0	CCP在[P1.2/ECI,P1.1/CCP0,P1.0/CCP1,P3.7/CCP2]
0	1	CCP在[P3.4/ECI_2,P3.5/CCP0_2,P3.6/CCP1_2,P3.7/CCP2_2]
1	0	CCP在[P2.4/ECI_3,P2.5/CCP0_3,P2.6/CCP1_3,P2.7/CCP2_3]
1	1	无效

SPI可在2个地方切换，由 SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI可在P1/P2之间来回切换
0	0	SPI在[P1.2/SS,P1.3/MOSI,P1.4/MISO,P1.5/SCLK]
0	1	SPI在[P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI在[P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	无效

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， 输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， 输出时钟频率 = MCLK / 4

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK是指主时钟频率。

STC15F408AD系列单片机在MCLKO/P5.4口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列)均在MCLKO/P5.4口对外输出主时钟。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,x00x
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000,x000

ADRJ：ADC转换结果调整

0：ADC_RES[7:0]存放高8位ADC结果，ADC_RESL[1:0]存放低2位ADC结果

1：ADC_RES[1:0]存放高2位ADC结果，ADC_RESL[7:0]存放低8位ADC结果

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

串口1的RxD管脚和TxD管脚可以在3组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1];
[RxD_2/P3.6, TxD_2/P3.7];
[RxD_3/P1.6, TxD_3/P1.7].

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 （系统时钟是指对主时钟进行分频后供给CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D转换的实际工作时钟）
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

STC MCU Limited

1.9.4 STC15F408AD系列单片机选型价格一览表

型号	工作电压（V）	Flash程序存储器(byte)	大容量SRAM字节	串行口	SPI仅有主机模式	普通定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	所有封装 SOP28 / SKDIP28 LQFP32		
																				封装价格 (RMB ¥)		
																				LQFP32	SOP28	SKDIP28
																				此系列的28pin单片机建议用STC15W408AS系列的相应封装的单片机取代		
STC15F408AD系列单片机选型价格一览表 特别提示：3路CCP/PCA/PWM还可当3路定时器使用																						
STC15F408AD	5.5-4.2	8K	512	1	有	2	3-ch	有	5	10位	1	5K	有	有	8级	有	是	有	是	¥2.9	¥2.8	¥3.0
IAP15F413AD	5.5-4.2	13K	512	1	有	2	3-ch	有	5	10位	1	IAP	有	有	8级	有	是	有	是	¥3.1	¥3.0	¥3.2
																				用户可将用户程序区的程序FLASH当EEPROM使用		
STC15L408AD系列单片机选型价格一览表																						
STC15L408AD	2.4-3.6	8K	512	1	有	2	3-ch	有	5	10位	1	5K	有	有	8级	有	是	有	是	¥2.9	¥2.8	¥3.0
IAP15L413AD	2.4-3.6	13K	512	1	有	2	3-ch	有	5	10位	1	IAP	有	有	8级	有	是	有	是	¥3.1	¥3.0	
																				用户可将用户程序区的程序FLASH当EEPROM使用		

提供客制化IC服务

如果要用28-pin单片机，建议用户选用SOP28封装；
如果要用32-pin单片机，建议用户选用LQFP32封装。

以上单价为10K起订
量小每片需加0.1元
以上价格运费由客户承担,零售10片起
如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

特别声明：以15F和15L开头且有SPI功能的芯片,只支持"SPI主机模式"，不支持"SPI从机模式"；以15W开头且有SPI功能的芯片，SPI主/从机模式均支持

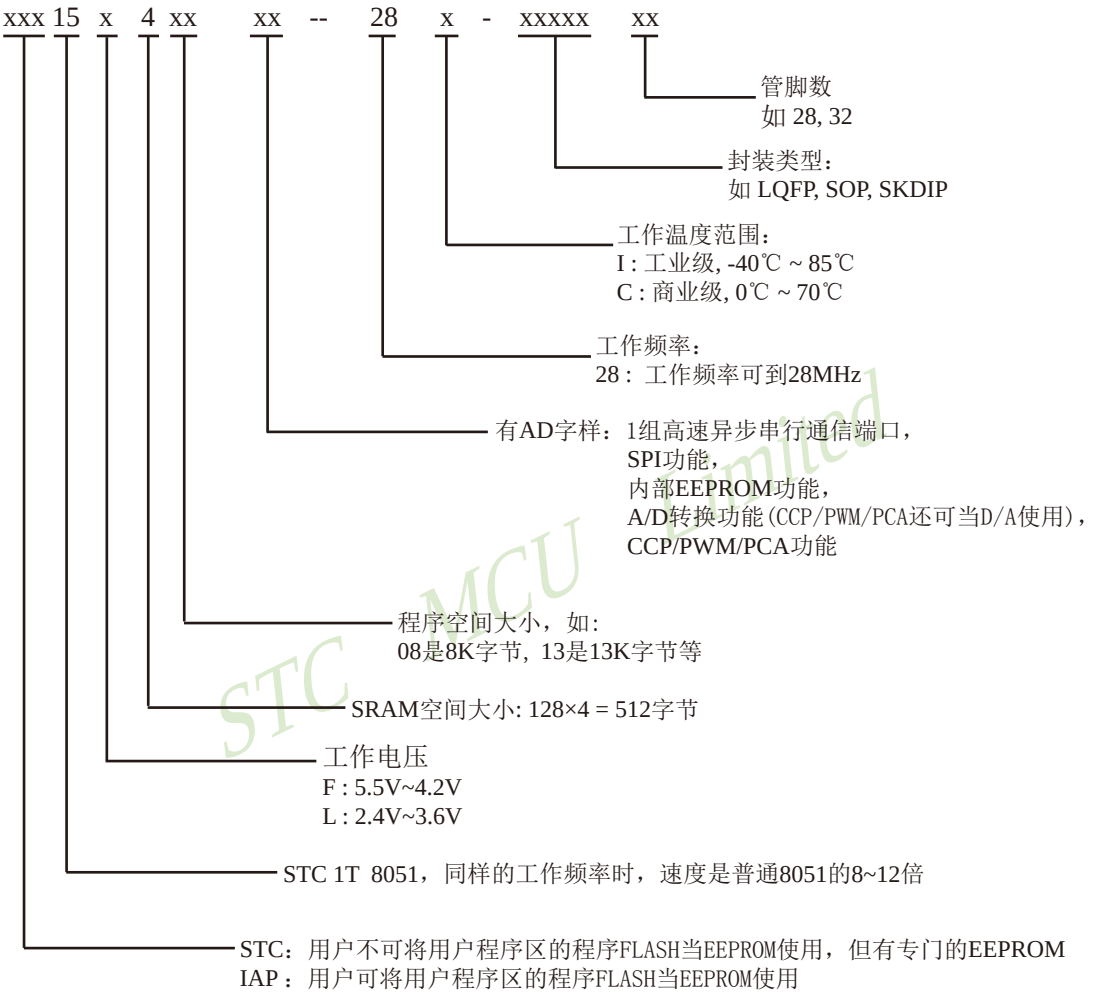
特别声明：以15L开头的芯片如需进入"掉电模式",进入"掉电模式"前必须启动掉电唤醒定时器<3uA>,不超过1秒要唤醒一次，以15F和15W开头的芯片则不需要

总结：STC15F408AD系列单片机有：2普通定时器/计数器(这2个普通定时器/计数器是指：T0和T2)，3路CCP/PWM/PCA(可再实现3个定时器使用)；掉电唤醒专用定时器；5个支持掉电唤醒的外部中断INT0/INT1/$\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$；1组高速异步串行通信端口；1组高速同步串行通信端口SPI；8路高速10位A/D转换器；1个数据指针DPTR等功能。STC15F408AD系列单片机没有外部数据总线。

因为程序区的最后7个字节单元被强制性的放入全球唯一ID号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少7个字节。

STC MCU Limited

1.9.5 STC15F408AD系列单片机命名规则



命名举例：

（1）STC15F408AD - 28I - SOP28 表示：
用户不可将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~4.2V，SRAM空间大小为512字节，程序空间大小为8K，有1组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为28。

（2）STC15L408AD - 28I - SOP28 表示：

用户不可将用户程序区的程序FLASH当EEPROM使用，但有专门的EEPROM，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为2.4V~3.6V，SRAM空间大小为512字节，程序空间大小为8K，有1组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为28。

（3）IAP15F413AD - 28I - SOP28 表示：

用户可将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为5.5V~4.2V，SRAM空间大小为512字节，程序空间大小为13K，有1组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为28。

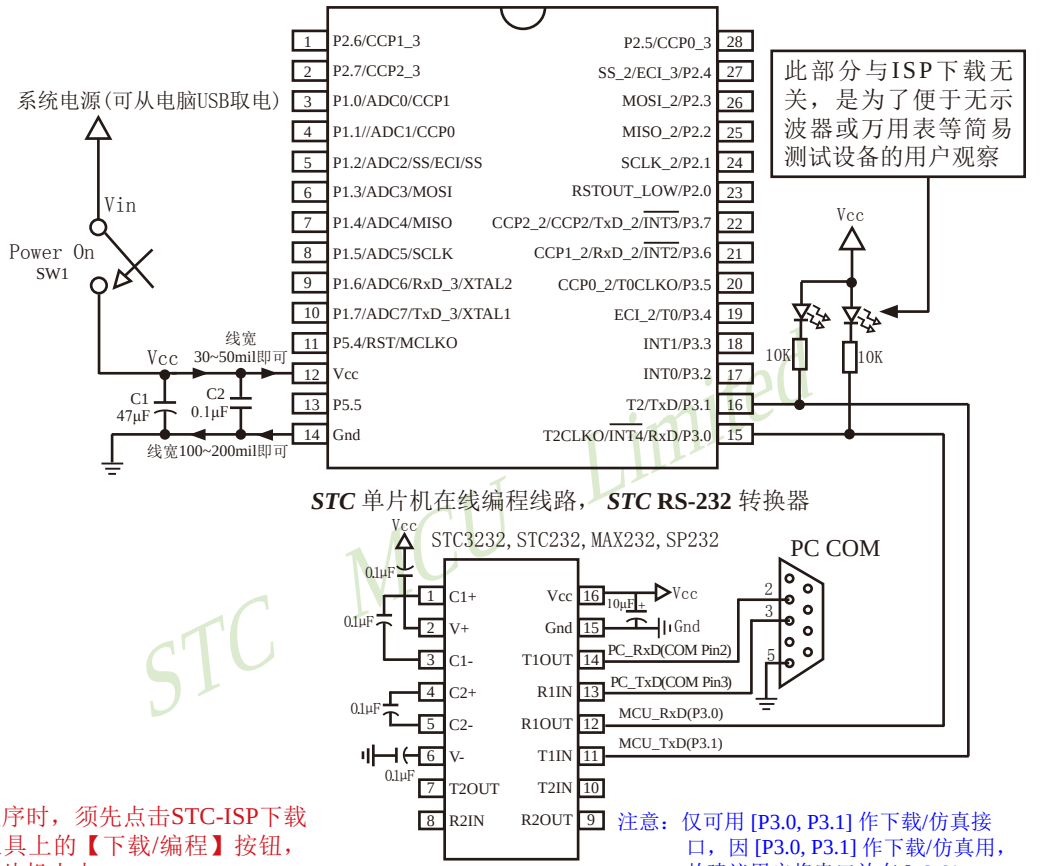
（4）IAP15L413AD - 28I - SOP28 表示：

用户可将用户程序区的程序FLASH当EEPROM使用，该单片机为1T 8051单片机，同样工作频率时，速度是普通8051的8~12倍，其工作电压为2.4V~3.6V，SRAM空间大小为512字节，程序空间大小为13K，有1组高速异步串行通信端口UART及SPI、内部EEPROM、A/D转换、CCP/PCA/PWM功能，工作频率可到28MHz，为工业级芯片，工作温度范围为-40℃ ~ 85℃，封装类型为SOP贴片封装，管脚数为28。

STC MCU Limited

1.9.6 STC15F408AD系列单片机在系统可编程(ISP)典型应用线路图

1.9.6.1 利用RS-232转换器的ISP下载编程典型应用线路图



烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

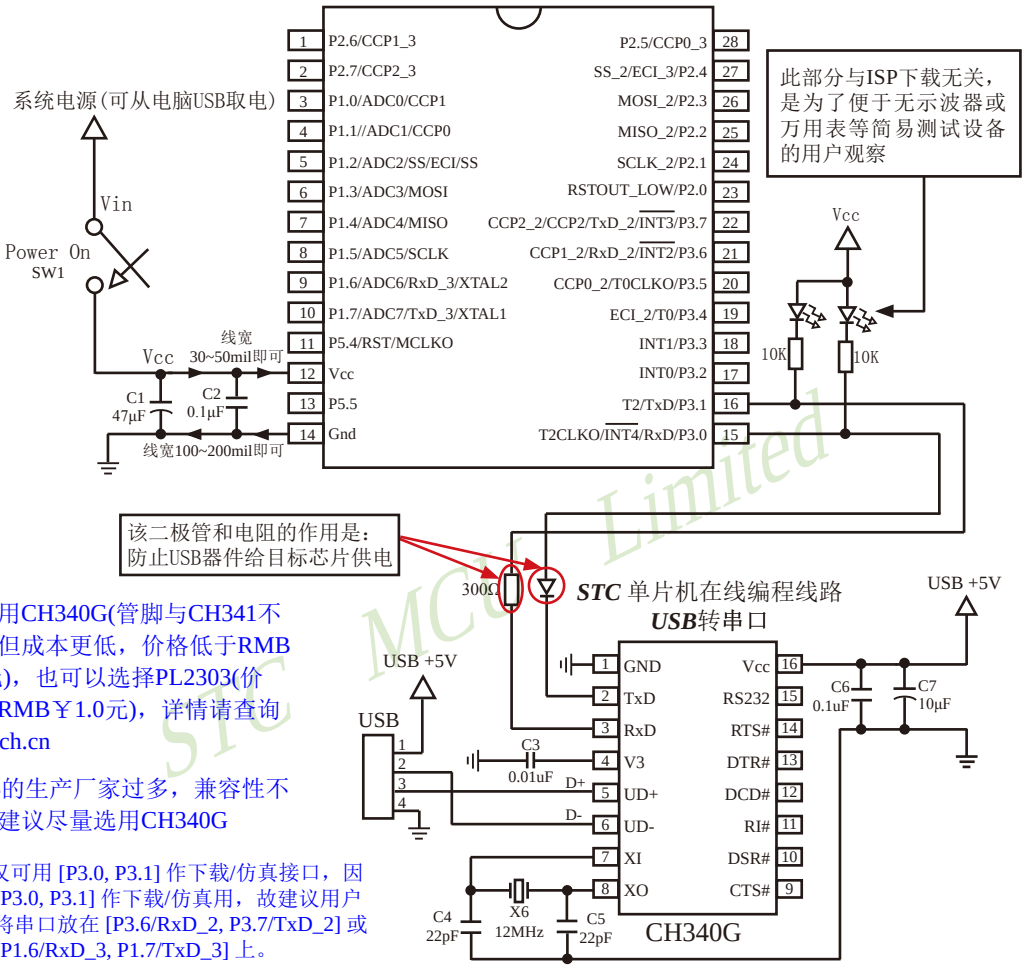
内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.9.6.2 利用USB转串口的ISP下载编程典型应用线路图



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

1.9.7 STC15F408AD系列单片机的管脚说明

管脚	管脚编号		说明	
	SOP28/ SKDIP28	LQFP32		
P0.0		29	标准I/O口 PORT0[0]	
P0.1		30	标准I/O口 PORT0[1]	
P0.2		31	标准I/O口 PORT0[2]	
P0.3		32	标准I/O口 PORT0[3]	
P1.0/ADC0/ CCP1	3	1	P1.0	标准I/O口 PORT1[0]
			ADC0	ADC 输入通道-0
			CCP1	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-1
P1.1/ADC1/ CCP0	4	2	P1.1	标准I/O口 PORT1[1]
			ADC1	ADC 输入通道-1
			CCP0	外部信号捕获(频率测量或当外部中断使用)、高速脉冲输出及脉宽调制输出通道-0
P1.2/ADC2/SS/ ECI	5	3	P1.2	标准I/O口 PORT1[2]
			ADC2	ADC 输入通道-2
			SS	SPI同步串行接口的从机选择信号
			ECI	CCP / PCA计数器的外部脉冲输入脚
P1.3/ADC3/ MOSI	6	4	P1.3	标准I/O口 PORT1[3]
			ADC3	ADC 输入通道-3
			MOSI	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P1.4/ADC4/ MISO	7	5	P1.4	标准I/O口 PORT1[4]
			ADC4	ADC 输入通道-4
			MISO	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P1.5/ADC5/ SCLK	8	6	P1.5	标准I/O口 PORT1[5]
			ADC5	ADC 输入通道-5
			SCLK	SPI同步串行接口的时钟信号
P1.6/ADC6/ RxD_3/XTAL2	9	7	P1.6	标准I/O口 PORT1[6]
			ADC6	ADC 输入通道-6
			RxD_3	串口数据接收端
			XTAL2	内部时钟电路反相放大器的输出端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚可浮空，此时XTAL2实际将XTAL1输入的时钟进行输出。
P1.7/ADC7/ TxD_3/XTAL1	10	8	P1.7	标准I/O口 PORT1[7]
			ADC7	ADC 输入通道-7
			TxD_3	串口数据发送端
			XTAL1	内部时钟电路反相放大器输入端，接外部晶振的其中一端。当直接使用外部时钟源时，此引脚是外部时钟源的输入端。

STC MCU Limited

管脚	管脚编号		说明	
	SOP28/ SKDIP28	LQFP32		
P2.0/ RSTOUT_LOW	23	21	P2.0	标准I/O口 PORT2[0]
			RSTOUT_LOW	上电后,输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P2.1/SCLK_2	24	22	P2.1	标准I/O口 PORT2[1]
			SCLK_2	SPI同步串行接口的时钟信号
P2.2/MISO_2	25	23	P2.2	标准I/O口 PORT2[2]
			MISO_2	SPI同步串行接口的主入从出(主器件的输入和从器件的输出)
P2.3/MOSI_2	26	24	P2.3	标准I/O口 PORT2[3]
			MOSI_2	SPI同步串行接口的主出从入(主器件的输出和从器件的输入)
P2.4/ECI_3/SS_2	27	25	P2.4	标准I/O口 PORT2[4]
			ECI_3	CCP / PCA计数器的外部脉冲输入脚
			SS_2	SPI同步串行接口的从机选择信号
P2.5/CCP0_3	28	26	P2.5	标准I/O口 PORT2[5]
			CCP0_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P2.6/CCP1_3	1	27	P2.6	标准I/O口 PORT2[6]
			CCP1_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
P2.7/CCP2_3	2	28	P2.7	标准I/O口 PORT2[7]
			CCP2_3	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
P3.0/RxD/$\overline{\text{INT4}}$ /T2CLKO	15	13	P3.0	标准I/O口 PORT3[0]
			RxD	串口1数据接收端
			$\overline{\text{INT4}}$	外部中断4, 只能下降沿中断, $\overline{\text{INT4}}$支持掉电唤醒
			T2CLKO	T2的时钟输出 可通过设置INT_CLKO[2]位/T2CLKO将该管脚配置为T2CLKO
P3.1/TxD/T2	16	14	P3.1	标准I/O口 PORT3[1]
			TxD	串口数据发送端
			T2	定时器/计数器2的外部输入
P3.2/INT0	17	15	P3.2	标准I/O口 PORT3[2]
			INT0	外部中断0，既可上升沿中断也可下降沿中断. 如果IT0(TCON.0)被置为1, INT0管脚仅为下降沿中断。如果IT0(TCON.0)被清0，INT0管脚既支持上升沿中断也支持下降沿中断。 INT0支持掉电唤醒。

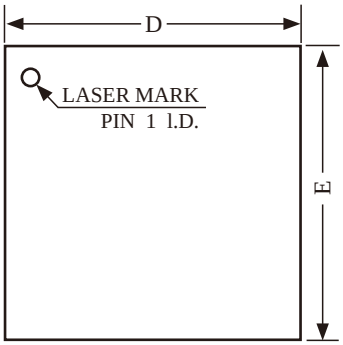
STC MCU Limited

管脚	管脚编号		说明	
	SOP28/ SKDIP28	LQFP32		
P3.3/INT1	18	16	P3.3	标准I/O口 PORT3[3]
			INT1	外部中断1，既可上升沿中断也可下降沿中断. 如果IT1(TCON.2)被置为1, INT1管脚仅为下降沿中断。如果IT1(TCON.2)被清0，INT1管脚既支持上升沿中断也支持下降沿中断。 INT1支持掉电唤醒。
P3.4/T0/ECI_2	19	17	P3.4	标准I/O口 PORT3[4]
			T0	定时器/计数器0的外部输入
			ECI_2	CCP/PCA计数器的外部脉冲输入脚
P3.5/T0CLKO/ CCP0_2	20	18	P3.5	标准I/O口 PORT3[5]
			T0CLKO	定时器/计数器0的时钟输出 可通过设置INT_CLKO[0]位/T0CLKO将该管脚配置为T0CLKO，也可对T0脚的外部时钟输入进行分频输出
			CCP0_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-0
P3.6/$\overline{\text{INT2}}$/RxD_2 /CCP1_2	21	19	P3.6	标准I/O口 PORT3[6]
			$\overline{\text{INT2}}$	外部中断2, 只能下降沿中断 $\overline{\text{INT2}}$支持掉电唤醒
			RxD_2	串口数据接收端
			CCP1_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-1
P3.7/$\overline{\text{INT3}}$/TxD_2/ CCP2/CCP2_2	22	20	P3.7	标准I/O口 PORT3[7]
			$\overline{\text{INT3}}$	外部中断3, 只能下降沿中断 $\overline{\text{INT3}}$支持掉电唤醒
			TxD_2	串口数据发送端
			CCP2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
			CCP2_2	外部信号捕获(频率测量或当外部中断使用）、高速脉冲输出及脉宽调制输出通道-2
P5.4/RST/ MCLKO	11	9	P5.4	标准I/O口 PORT5[4]
			RST	复位脚(高电平复位)
			MCLKO	主时钟输出：输出的频率可为MCLK/1, MCLK/2, MCLK/4 (MCLK是指主时钟频率)。 主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK指主时钟频率。
P5.5	13	11	标准I/O口 PORT5[5]	
Vcc	12	10	电源正极	
Gnd	14	12	电源负极，接地	

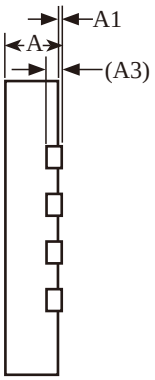
STC MCU Limited

1.10 STC15系列单片机封装尺寸图

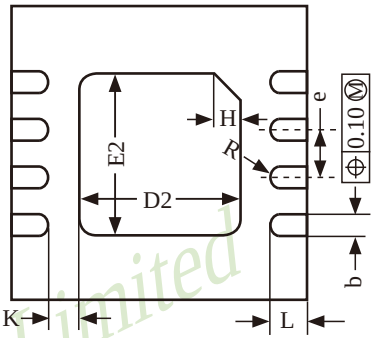
1.10.1 DFN8封装尺寸图(不推荐，推荐选择SOP8封装)



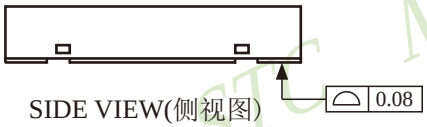
TOP VIEW(顶视图)



SIDE VIEW(侧视图)



BOTTOM VIEW(底视图)



SIDE VIEW(侧视图)

一般尺寸			
测量单位 = 毫米(MILLIMETER)			
符号	MIN.	NOM.	MAX.
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3		0.20REF	
b	0.25	0.30	0.35
D	3.90	4.00	4.10
E	3.90	4.00	4.10
D2	2.10	2.20	2.30
E2	2.10	2.20	2.30
e	0.55	0.65	0.75
H		0.35REF	
K		0.35REF	
L	0.45	0.55	0.65
R	0.13	-	-

Note:

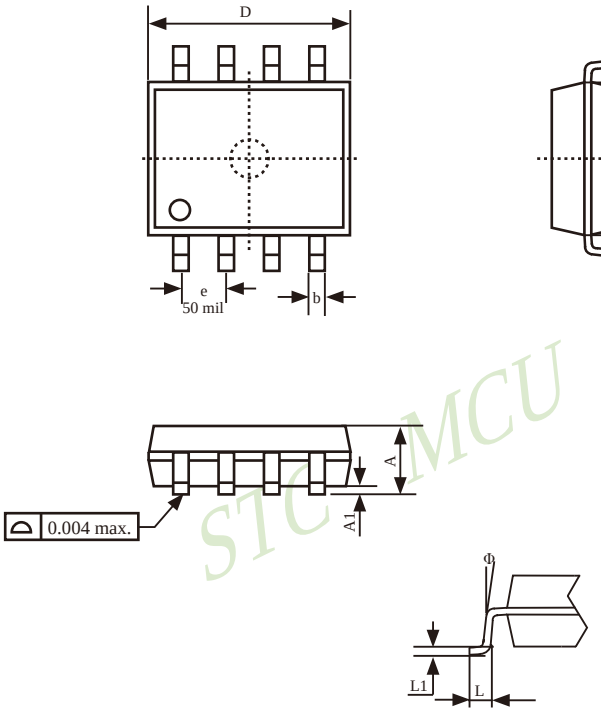
All dimensions do not include mold flash or protrusions

STC MCU Limited

1.10.2 SOP8封装尺寸图

SOP8 封装尺寸图

8-PIN SMALL OUTLINE PACKAGE (SOP-8)
Dimensions in Inches



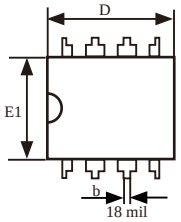
STC MCU Limited

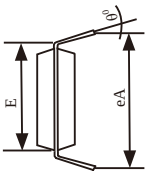
一般尺寸			
(测量单位 = INCH)			
符号	MIN.	NOM.	MAX.
A	0.053	-	0.069
A1	0.004	-	0.010
b	-	0.016	-
D	0.189	-	0.196
E	0.228	-	0.244
E1	0.150	-	0.157
e	0.050		
L	0.016	-	0.050
L1	0.008		
Φ	0^0	-	8^0

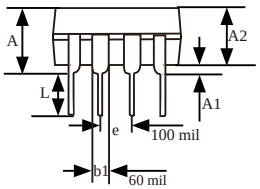
UNIT: INCH,　1 inch = 1000 mil

1.10.3 DIP8封装尺寸图

8-Pin Plastic Dual Inline Package (DIP8)
Dimensions in Inches







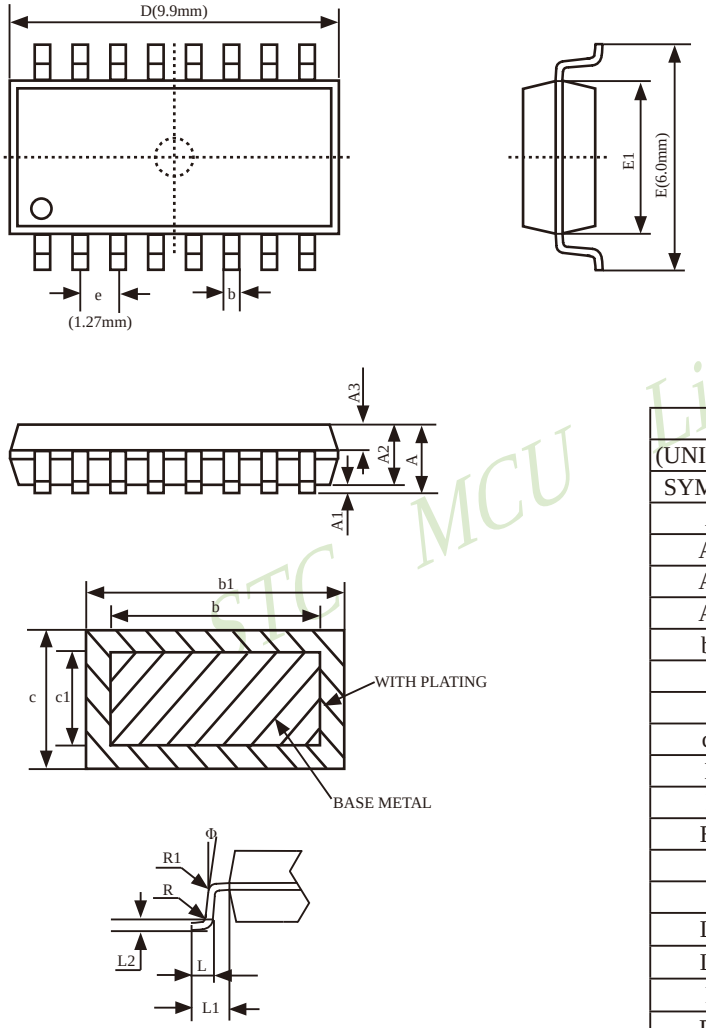
一般尺寸			
(测量单位 = INCH)			
符号	MIN.	NOM.	MAX.
A	-	-	0.210
A1	0.015	-	-
A2	0.125	0.130	0.135
b	-	0.018	-
b1	-	0.060	-
D	0.355	0.365	0.400
E	-	0.300	-
E1	0.245	0.250	0.255
e	-	0.100	-
L	0.115	0.130	0.150
θ^0	0	7	15
eA	0.335	0.355	0.375

UNIT: INCH, 1 inch = 1000 mil

STC MCU limited

1.10.4 SOP16封装尺寸图

16-PIN SMALL OUTLINE PACKAGE (SOP16)

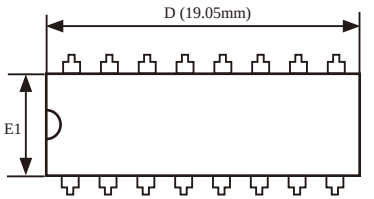


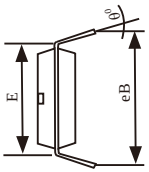
COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLMETER)			
SYMBOL	MIN	NOM	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b1	0.36	-	0.49
b	0.35	0.40	0.45
c	0.16	-	0.25
c1	0.15	0.20	0.25
D	9.80	9.90	10.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e		1.27	
L	0.45	0.60	0.80
L1		1.04	
L2		0.25	
R	0.07	-	-
R1	0.07	-	-
Φ	6^0	8^0	10^0

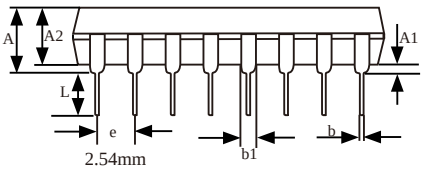
STC MCU Limited

1.10.5 DIP16封装尺寸图

16-Pin Plastic Dual Inline Package (DIP16)
Dimensions in Inches and Millmeters





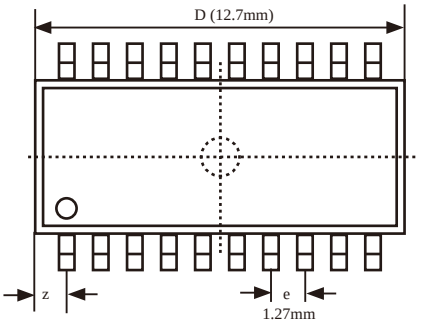


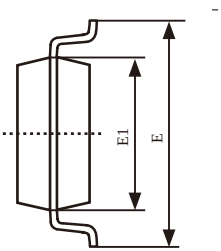
COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLMETER)			
SYMBOL	MIN	NOM	MAX
A	-	-	4.80
A1	0.50	-	-
A2	3.10	3.30	3.50
b	0.38	-	0.55
b1	0.38	0.46	0.51
D	18.95	19.05	19.15
E	7.62	7.87	8.25
E1	6.25	6.35	6.45
e	2.54		
eB	7.62	8.80	10.90
L	2.92	3.30	3.81
θ^{0S}	0	7	15

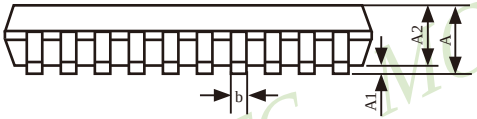
STC MCU Limited

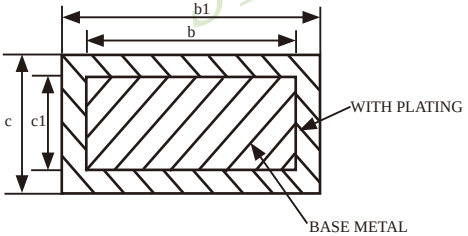
1.10.6 SOP20封装尺寸图

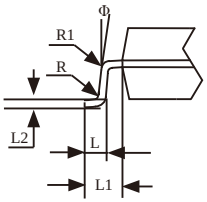
20-Pin Small Outline Package (SOP20)
Dimensions in Inches and (Millimeters)











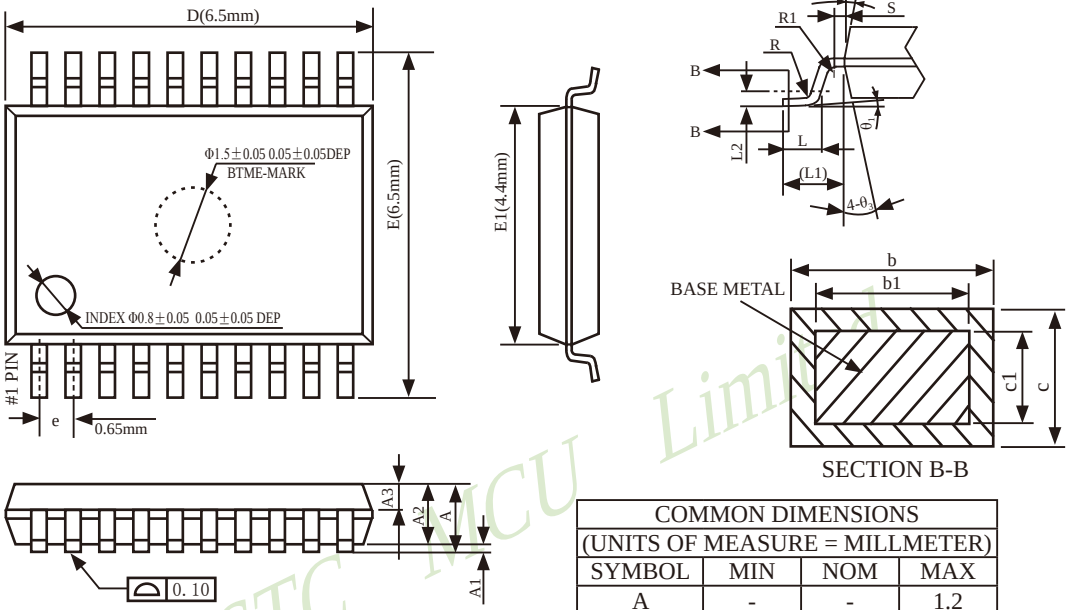
一般尺寸			
(测量单位 = MILLMETER/ mm)			
符号	MIN.	NOM.	MAX.
A	2.465	2.515	2.565
A1	0.100	0.150	0.200
A2	2.100	2.300	2.500
b1	0.366	0.426	0.486
b	0.356	0.406	0.456
c	0.234	-	0.274
c1	-	0.254	-
D	12.500	12.700	12.900
E	10.206	10.306	10.406
E1	7.450	7.500	7.550
e	1.27		
L	0.800	0.864	0.900
L1	1.303	1.403	1.503
L2	-	0.274	-
R	-	0.300	-
R1	-	0.200	-
Φ	0^0	-	10^0
z	-	0.660	-

STC MCU Limited

1.10.7 TSSOP20封装尺寸图

20-Pin Plastic Thin Shrink Small Outline Package (TSSOP20)

Dimensions in Millimeters



STC MCU Limited

COMMON DIMENSIONS (UNITS OF MEASURE = MILLMETER)			
SYMBOL	MIN	NOM	MAX
A	-	-	1.2
A1	0.05	-	0.15
A2	0.90	1.00	1.05
A3	0.34	0.44	0.54
b	0.20	-	0.28
b1	0.20	-	0.24
c	0.10	-	0.19
c1	0.10	0.13	0.15
D	6.40	6.50	6.60
E	6.20	6.50	6.60
E1	4.30	4.40	4.50
e	0.65BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R	0.09	-	-
R1	0.09	-	-
S	0.20	-	-
θ_1	0^0	-	8^0
θ_2	10^0	12^0	14^0
θ_3	10^0	12^0	14^0

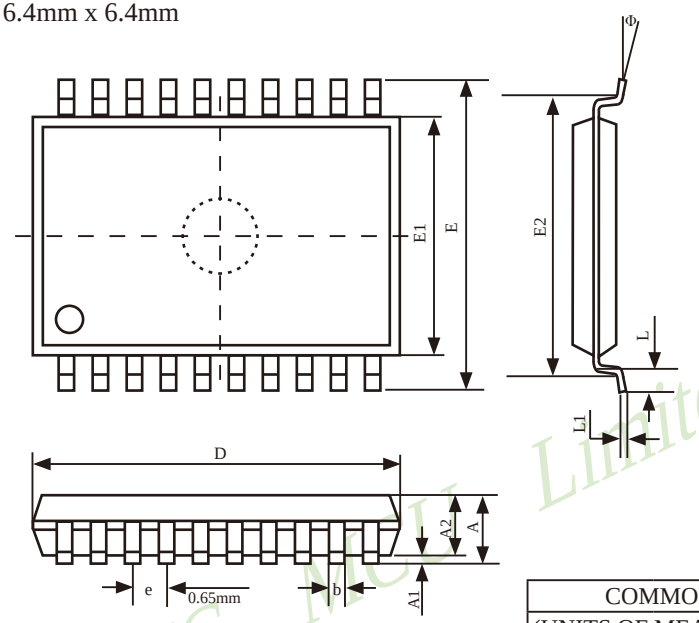
△2

NOTES:
ALL DIMENSIONS REFER TO JEDEC STANDARD MO-153 AC
DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.

1.10.8 LSSOP20封装尺寸图

20-Pin Plastic Shrink Small Outline Package (LSSOP20)

LSSOP20, 6.4mm x 6.4mm

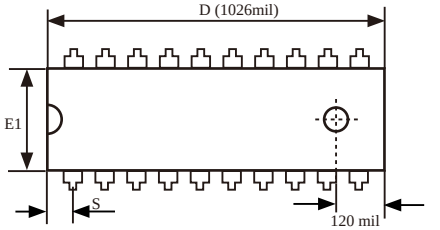


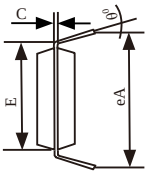
STC MCU Limited

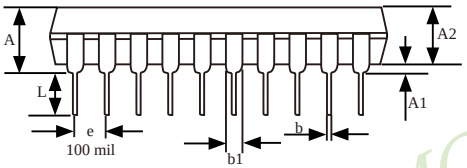
COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLMETER)			
SYMBOL	MIN	NOM	MAX
A	-	-	1.85
A1	0.05	-	-
A2	1.40	1.50	1.60
b	0.17	0.22	0.32
D	6.40	6.50	6.60
E	6.20	6.40	6.60
E1	4.30	4.40	4.50
E2	-	5.72	-
e	0.57	0.65	0.73
L	0.30	0.50	0.70
L1	0.1	0.15	0.25
Φ	0^0	-	8^0

1.10.9 DIP20封装尺寸图

20-Pin Plastic Dual Inline Package (DIP20)
Dimensions in Inches







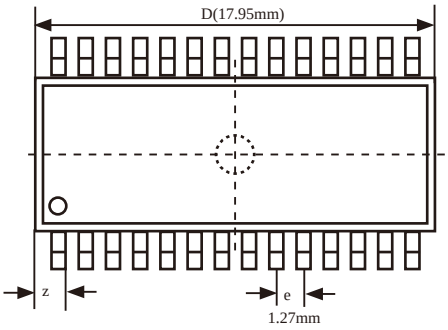
一般尺寸 (测量单位 = INCH)			
符号	MIN.	NOM.	MAX.
A	-	-	0.175
A1	0.015	-	-
A2	0.125	0.13	0.135
b	0.016	0.018	0.020
b1	0.058	0.060	0.064
C	0.008	0.010	0.11
D	1.012	1.026	1.040
E	0.290	0.300	0.310
E1	0.245	0.250	0.255
e	0.090	0.100	0.110
L	0.120	0.130	0.140
θ^0	0	-	15
eA	0.355	0.355	0.375
S	-	-	0.075

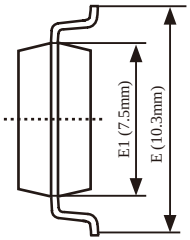
UNIT: INCH,　1 inch = 1000 mil

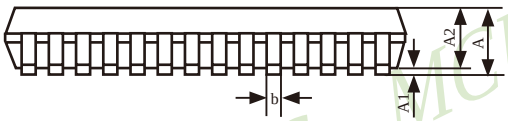
STC MCU Limited

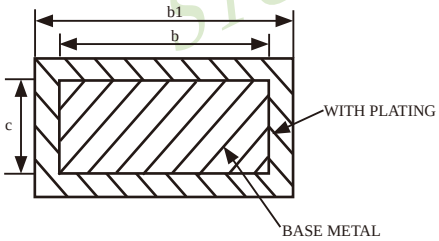
1.10.10 SOP28封装尺寸图

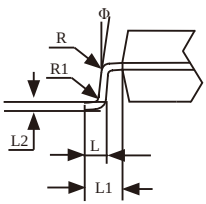
28-Pin Small Outline Package (SOP28)
Dimensions in Millimeters











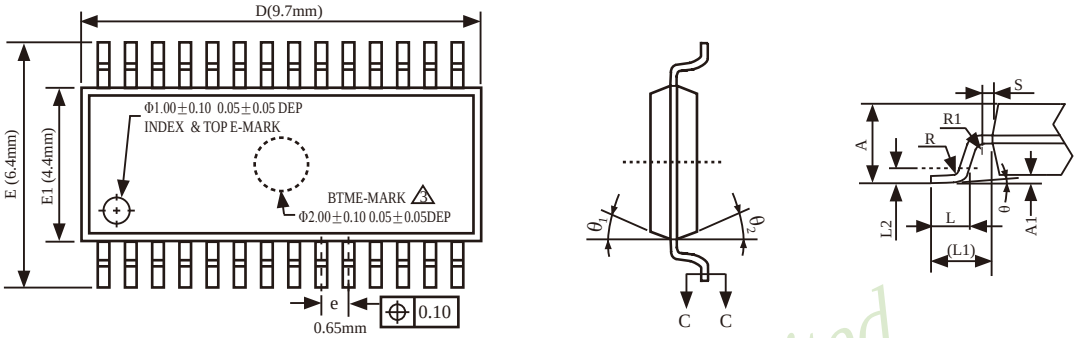
一般尺寸			
(测量单位 = MILLMETER / mm)			
符号	MIN.	NOM.	MAX.
A	2.465	2.515	2.565
A1	0.100	0.150	0.200
A2	2.100	2.300	2.500
b	0.356	0.406	0.456
b1	0.366	0.426	0.486
c	-	0.254	-
D	17.750	17.950	18.150
E	10.100	10.300	10.500
E1	7.424	7.500	7.624
e	1.27		
L	0.764	0.864	0.964
L1	1.303	1.403	1.503
L2	-	0.274	-
R	-	0.200	-
R1	-	0.300	-
Φ	0^0	-	10^0
z	-	0.745	-

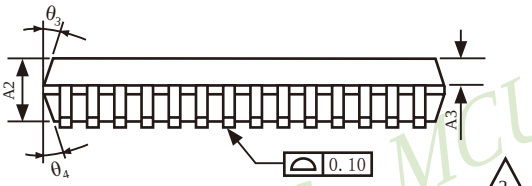
STC MCU Limited

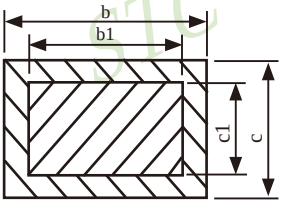
1.10.11 TSSOP28封装尺寸图

28-Pin Plastic Thin Shrink Small Outline Package (TSSOP28)

Dimensions in Millimeters







SECTION C-C

NORMAL PLATING

一般尺寸			
(测量单位 = MILLMETER / mm)			
符号	MIN.	NOM.	MAX.
A	-	-	1.20
A1	0.05	-	0.15
A2	0.90	1.00	1.05
A3	0.34	0.44	0.54
b	0.20	-	0.29
b1	0.19	0.22	0.25
c	0.13	-	0.18
c1	0.12	0.13	0.14
D	9.60	9.70	9.80
E	6.20	6.40	6.60
E1	4.30	4.40	4.50
e	0.55	0.65	0.75
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R	0.09	-	-
R1	0.09	-	-
S	0.20	-	-
θ	0^0	-	8^0
θ_1	10^0	12^0	14^0
θ_2	10^0	12^0	14^0
θ_3	10^0	12^0	14^0
θ_4	10^0	12^0	14^0

NOTES:

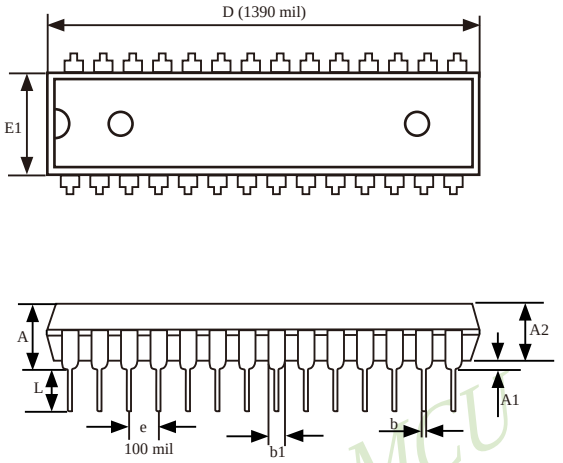
ALL DIMENSIONS REFER TO JEDEC STANDARD MO-153 AE

DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.

STC MCU Limited

1.10.12 SKDIP28封装尺寸图

28-Pin Plastic Dual-In-line Package (SKDIP28)
Dimensions in Inches



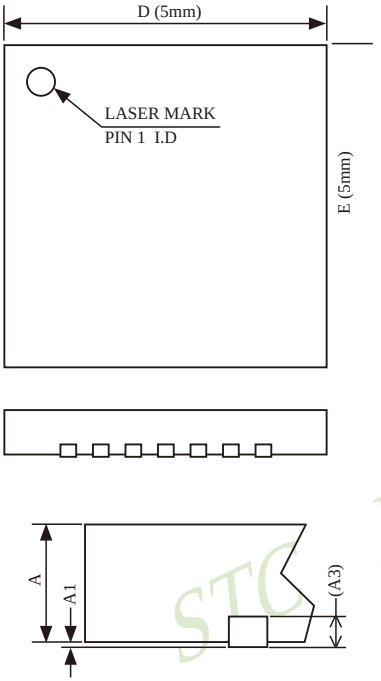
一般尺寸 (测量单位 = INCH)			
符号	MIN.	NOM.	MAX.
A	-	-	0.210
A1	0.015	-	-
A2	0.125	0.13	0.135
b	-	0.018	-
b1	-	0.060	-
D	1.385	1.390	1.40
E	-	0.310	-
E1	0.283	0.288	0.293
e	-	0.100	-
L	0.115	0.130	0.150
θ^0	0	7	15
eA	0.330	0.350	0.370

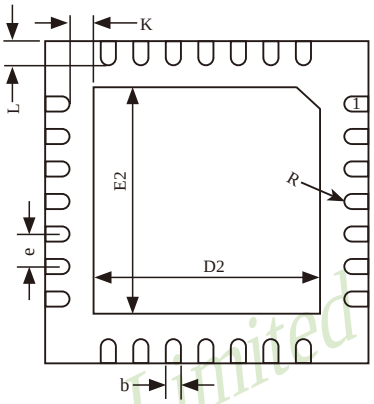
UNIT: INCH, 1 inch = 1000 mil

STC MCU Limited

1.10.13 QFN28封装尺寸图

QFN28 OUTLINE PACKAGE





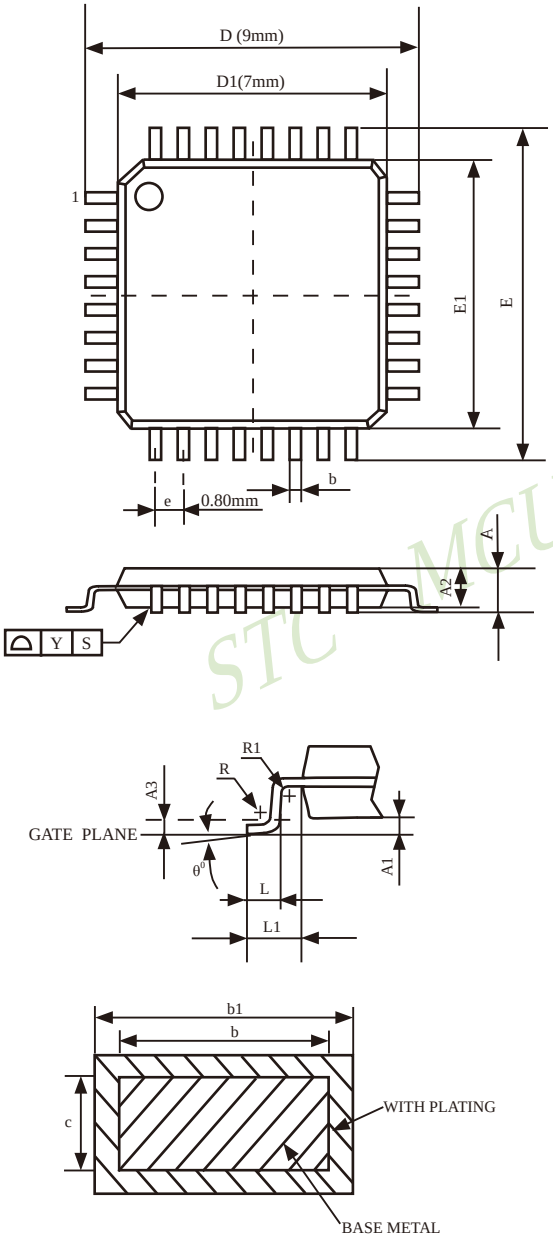
STC MCU Limited

一般尺寸			
(测量单位 = MILLMETER / mm)			
符号	MIN.	NOM.	MAX.
A	0.70	0.75	0.80
A1	0	0.02	0.05
A3		0.20REF	
b	0.20	0.25	0.30
D	4.90	5.00	5.10
E	4.90	5.00	5.10
D2	3.35	3.50	3.65
E2	3.35	3.50	3.65
e	0.40	0.50	0.60
K	0.20	-	-
L	0.30	0.40	0.50
R	0.09	-	-

NOTES:
ALL DIMENSIONS REFER TO JEDEC STANDARD MO-220 WHHD-3

1.10.14 LQFP32封装尺寸图

LQFP32 OUTLINE PACKAGE



VARIATIONS (ALL DIMENSIONS SHOWN IN MM)

SYMBOLS	MIN.	NOM	MAX.
A	1.45	1.55	1.65
A1	0.01	-	0.21
A2	1.35	1.40	1.45
A3	-	0.254	-
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e		0.80	
b	0.3	0.35	0.4
b1	0.31	0.37	0.43
c	-	0.127	-
L	0.43	-	0.71
L1	0.90	1.00	1.10
R	0.1	-	0.25
R1	0.1	-	-
θ^0	0^0	-	10^0

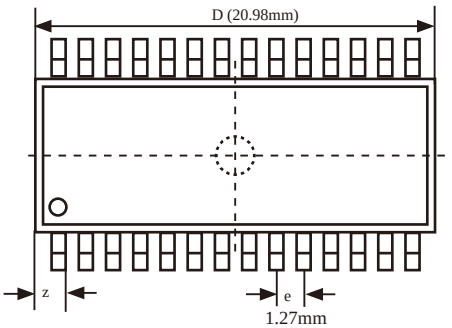
NOTES:
1. All dimensions are in mm
2. Dim D1 AND E1 does not include plastic flash.
Flash:Plastic residual around body edge after de junk/singulation
3. Dim b does not include dambar protrusion/ intrusion.
4. Plating thickness 0.05~0.015 mm.

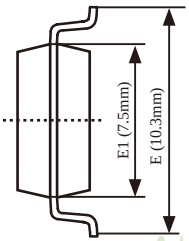
STC MCU Limited

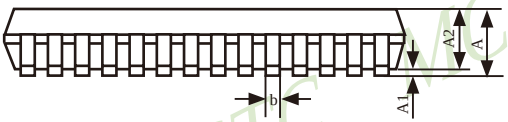
1.10.15 SOP32封装尺寸图

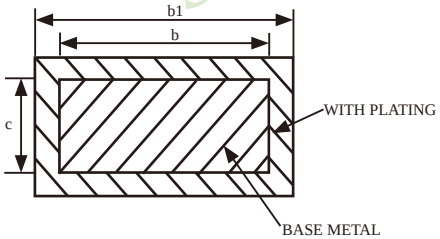
SOP32 封装尺寸图(STC15系列暂未生产SOP32封装，请选择LQFP32封装)

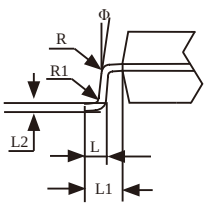
32-Pin Small Outline Package (SOP32)
Dimensions in Millimeters









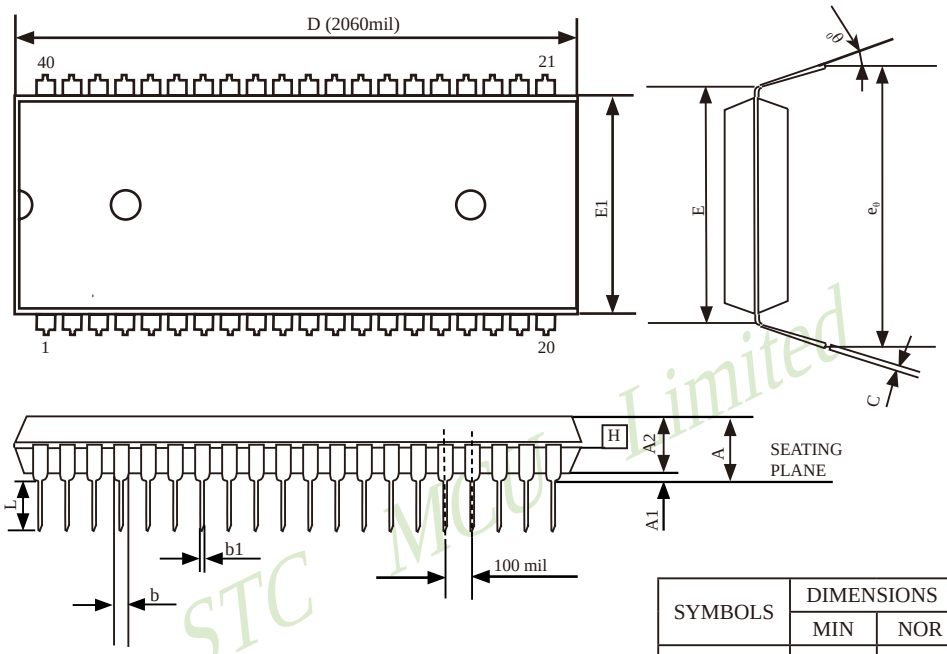


一般尺寸			
(测量单位 = MILLMETER /mm)			
SYMBOL	MIN	NOM	MAX
A	2.465	2.515	2.565
A1	0.100	0.150	0.200
A2	2.100	2.300	2.500
b	0.356	0.406	0.456
b1	0.366	0.426	0.486
c	-	0.254	-
D	20.88	20.98	21.08
E	10.100	10.300	10.500
E1	7.424	7.500	7.624
e		1.27	
L	0.700	0.800	0.900
L1	1.303	1.403	1.503
L2	-	0.274	-
R	-	0.200	-
R1	-	0.300	-
Φ	0^0	-	10^0
z	-	0.745	-

STC MCU Limited

1.10.16 PDIP40封装尺寸图

PDIP40 OUTLINE PACKAGE



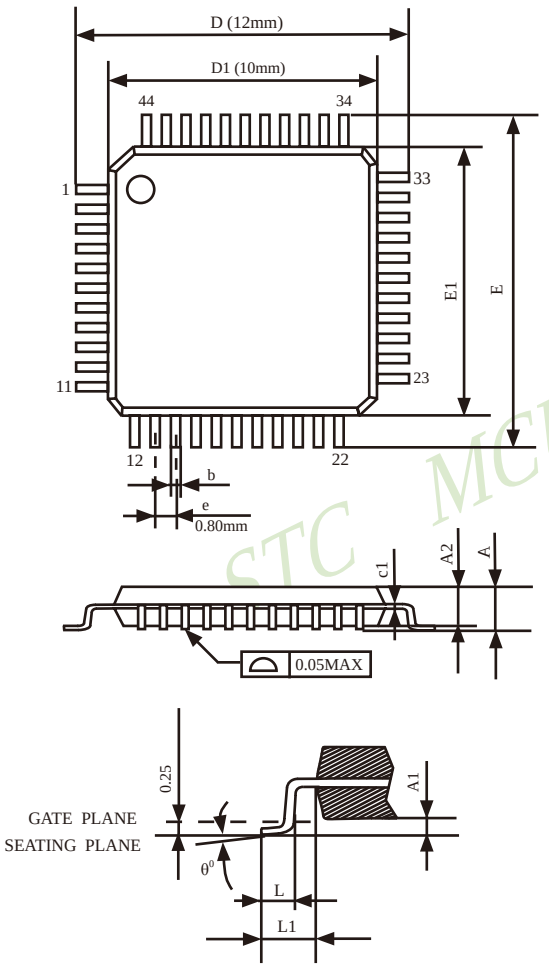
SYMBOLS	DIMENSIONS IN INCH		
	MIN	NOR	MAX
A	-	-	0.190
A1	0.015	-	0.020
A2	0.15	0.155	0.160
C	0.008	-	0.015
D	2.025	2.060	2.070
E	0.600 BSC		
E1	0.540	0.545	0.550
L	0.120	0.130	0.140
b1	0.015	-	0.021
b	0.045	-	0.067
e_θ	0.630	0.650	0.690
θ	0	7	15

UNIT: INCH　1 inch = 1000mil

STC MCU Limited

1.10.17 LQFP44封装尺寸图

LQFP44 OUTLINE PACKAGE



VARIATIONS (ALL DIMENSIONS SHOWN IN MM

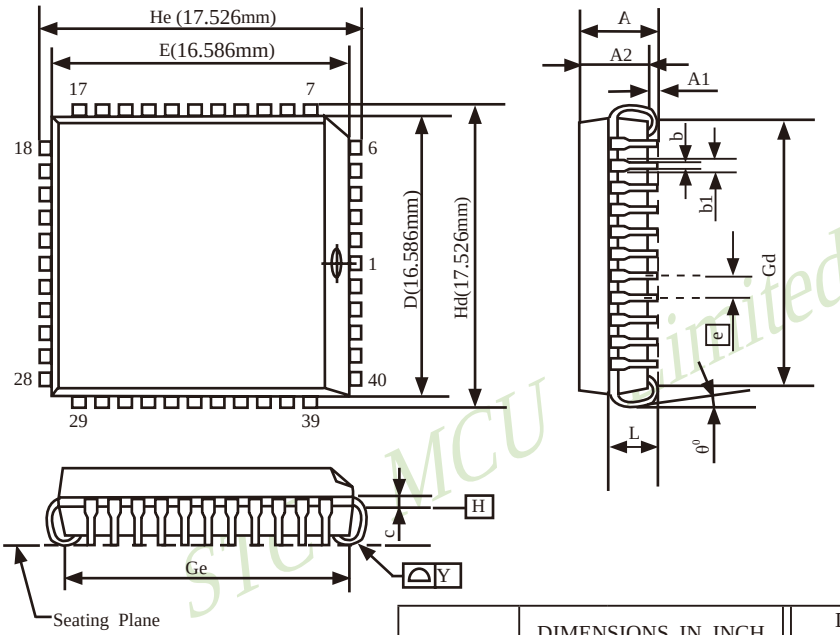
SYMBOLS	MIN.	NOM	MAX.
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
c1	0.09	-	0.16
D		12.00	
D1		10.00	
E		12.00	
E1		10.00	
e		0.80	
b(w/o plating)	0.25	0.30	0.35
L	0.45	0.60	0.75
L1		1.00REF	
θ^0	0^0	3.5^0	7^0

STC MCU Limited

1.10.18 PLCC44封装尺寸图

PLCC44 封装尺寸图(STC15系列暂未生产PLCC44封装，请选择LQFP44封装)

PLCC44 OUTLINE PACKAGE



SYMBOLS	DIMENSIONS IN INCH			DIMENSIONS IN MILLMETERS		
	MIN	NOM	MAX	MIN	NOM	MAX
A	0.165	-	0.180	4.191	-	4.572
A1	0.020	-	-	0.508	-	-
A2	0.147	-	0.158	3.734	-	4.013
b1	0.026	0.028	0.032	0.660	0.711	0.813
b	0.013	0.017	0.021	0.330	0.432	0.533
c	0.007	0.010	0.0013	0.178	0.254	0.330
D	0.650	0.653	0.656	16.510	16.586	16.662
E	0.650	0.653	0.656	16.510	16.586	16.662
e	0.050BSC			1.270BSC		
Gd	0.590	0.610	0.630	14.986	15.494	16.002
Ge	0.590	0.610	0.630	14.986	15.494	16.002
Hd	0.685	0.690	0.695	17.399	17.526	17.653
He	0.685	0.690	0.695	17.399	17.526	17.653
L	0.100	-	0.112	2.540	-	2.845
Y	-	-	0.004	-	-	0.102

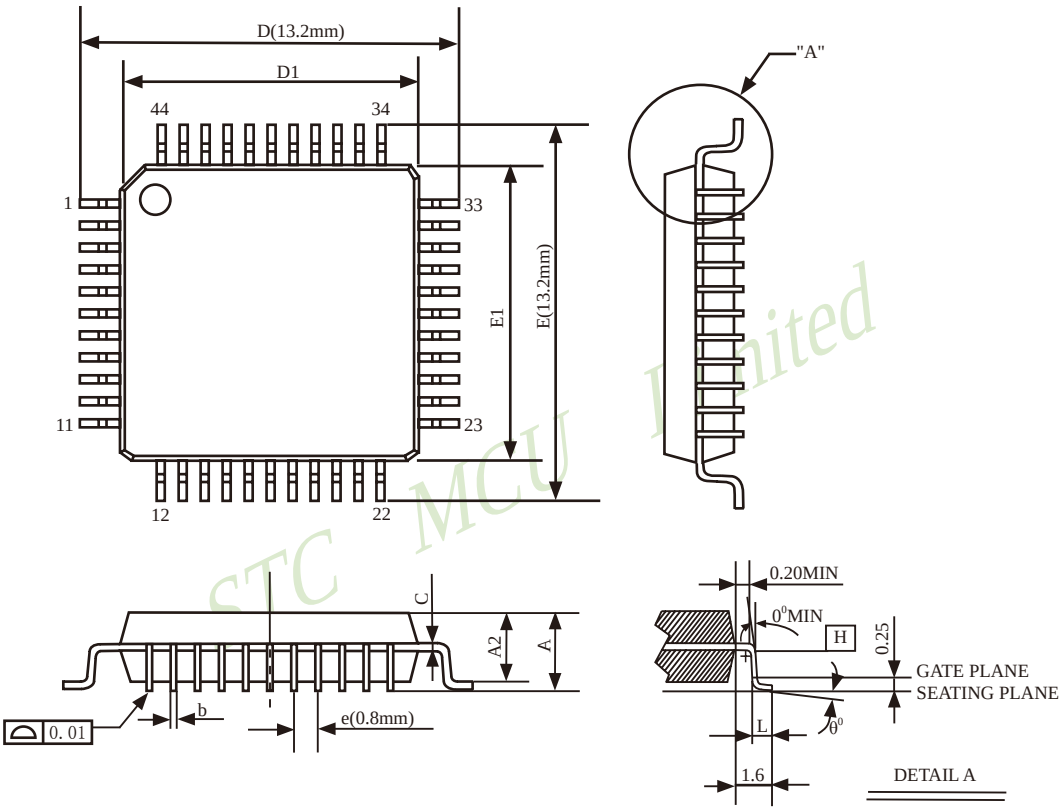
1 inch = 1000 mil

STC MCU Limited

1.10.19 PQFP44封装尺寸图

PQFP44 封装尺寸图(STC15系列现未生产PQFP44的封装，建议选择LQFP44封装)

PQFP44 OUTLINE PACKAGE



SYMBOLS	MIN.	NOM	MAX.
A	–	–	2.70
A1	0.25	–	0.50
A2	1.80	2.00	2.20
b(w/o plating)	0.25	0.30	0.35
D	13.00	13.20	13.40
D1	9.9	10.00	10.10
E	13.00	13.20	13.40
E1	9.9	10.00	10.10
L	0.73	0.88	0.93
e	0.80 BSC.		
θ°	0	–	7
C	0.1	0.15	0.2

UNIT:mm

NOTES:
1.JEDEC OUTLINE:M0-108 AA-1

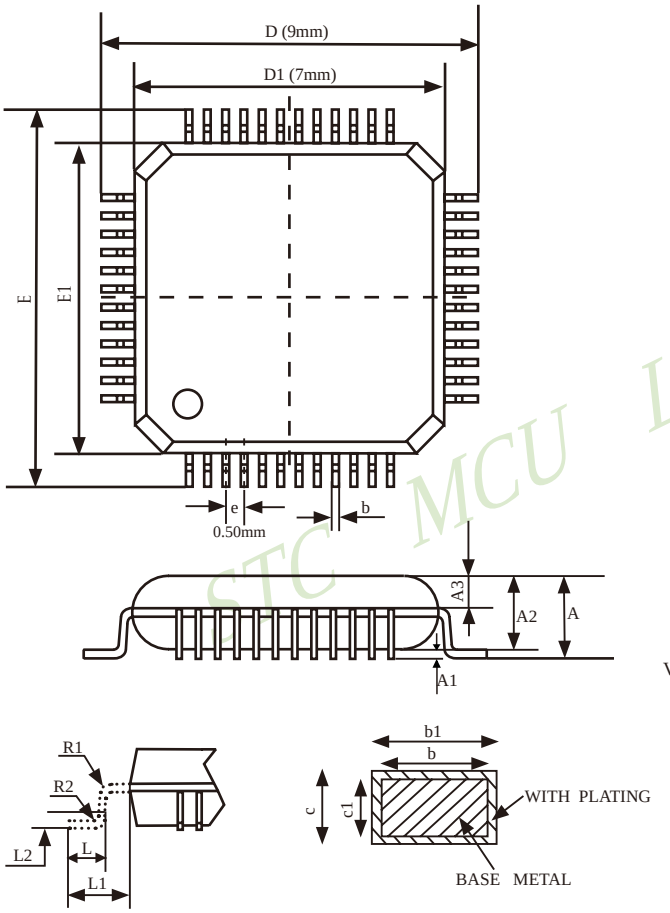
2.DATUM PLANE H IS LOCATED AT THE BOTTOM OF THE MOLD PARTING LINE COINCIDENT WITH WHERE THE LAED EXITS THE BODY.

3.DIMENSIONS D1 AND E1 D0 NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25mm PER SIDE. DIMENSIONS D1 AND E1 D0 INCLUDE MOLD MISMATCH AND ARE DETRMINED AT DATUM PLANE H.

4.DIMENSION b DOES NOT INCLUDE DAMBAR PROTRUSION.

1.10.20 LQFP48封装尺寸图

LQFP48 OUTLINE PACKAGE



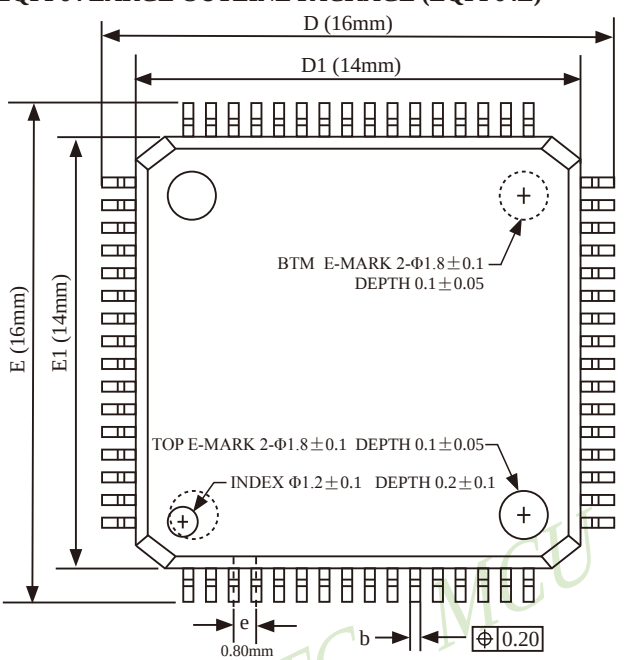
SYMBOL	MIN	NOM	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.50		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-

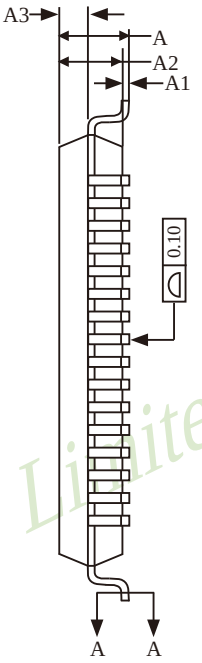
VARIATIONS (ALL DIMENSIONS SHOWN IN MM

STC MCU Limited

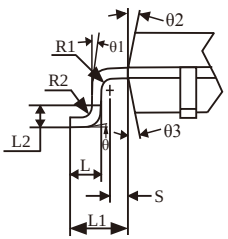
1.10.21 LQFP64封装尺寸图

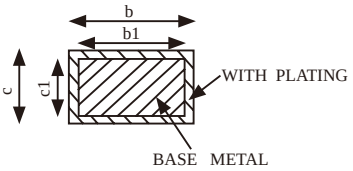
LQFP64 LARGE OUTLINE PACKAGE (LQFP64L)





一般尺寸 (测量单位 = MILLMETER /mm)			
SYMBOL	MIN	NOM	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.31	-	0.44
b1	0.30	0.35	0.40
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	15.80	16.00	16.20
D1	13.90	14.00	14.10
E	15.8	16.00	16.20
E1	13.90	14.00	14.10
e	0.70	0.80	0.90
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-
θ	0^0	3.5^0	7^0
θ1	0^0	-	-
θ2	11^0	12^0	13^0
θ3	11^0	12^0	13^0



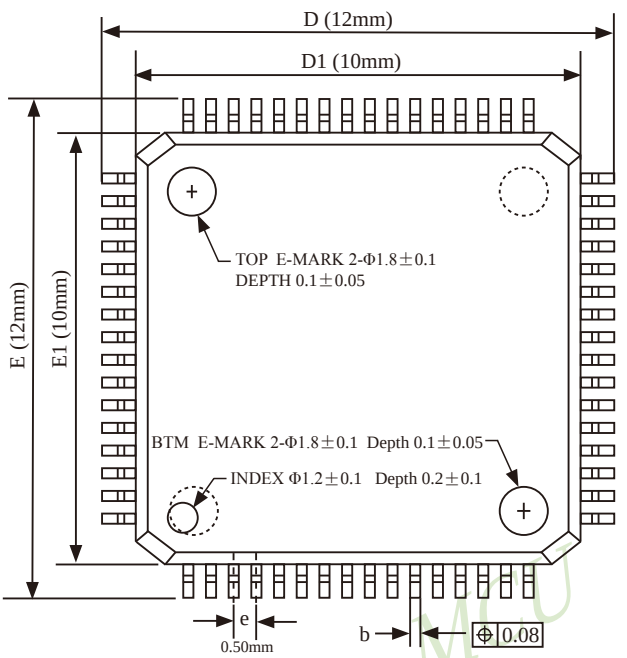


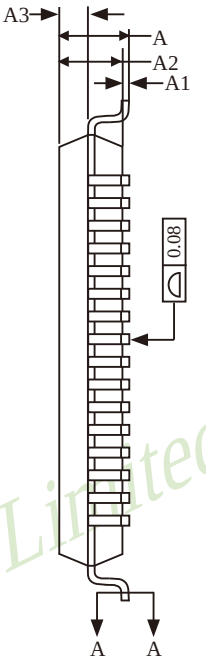
A-A的截面图

NOTES:
ALL DIMENSIONS MEET JEDEC STANDARD MS-026 BEB DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.

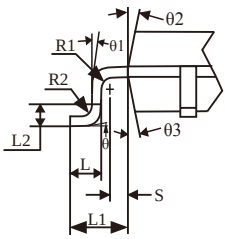
STC MCU Limited

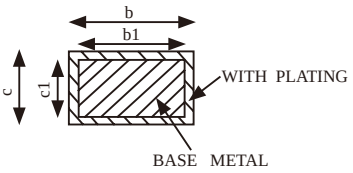
LQFP64 SMALL OUTLINE PACKAGE (LQFP64S)





一般尺寸			
(测量单位 = MILLMETER /mm)			
SYMBOL	MIN	NOM	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	11.80	12.00	12.20
D1	9.90	10.00	10.10
E	11.80	12.00	12.20
E1	9.90	10.00	10.10
e	0.50BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-
θ	0^0	3.5^0	7^0
θ1	0^0	-	-
θ2	11^0	12^0	13^0
θ3	11^0	12^0	13^0





A-A的截面图

NOTES:
ALL DIMENSIONS MEET JEDEC STANDARD MS-026 BEB DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.

STC MCU Limited

1.11 特殊外围设备(CCP/SPI, 串口1/2/3/4)在不同口间进行切换

CCP：是英文单词的缩写
Capture(捕获),Compare(比较),PWM(脉宽调制)

STC15W4K60S4的特殊外围设备CCP/PWM、SPI、串口1、串口2、串口3、串口4可以在多个口之间进行任意切换。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000

CCP可在3个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP可在P1/P2/P3之间来回切换
0	0	CCP在[P1.2/ECI,P1.1/CCP0,P1.0/CCP1]
0	1	CCP在[P3.4/ECI_2,P3.5/CCP0_2,P3.6/CCP1_2]
1	0	CCP在[P2.4/ECI_3,P2.5/CCP0_3,P2.6/CCP1_3]
1	1	无效

PWM2/PWM3/PWM4/PWM5/PWMFLT可在2个地方切换，由 PWM2345_S 控制位来选择	
PWM2345_S	切换PWM2/PWM3/PWM4/PWM5/PWMFLT管脚
0	PWM2/PWM3/PWM4/PWM5/PWMFLT在[P3.7/PWM2, P2.1/PWM3, P2.2/PWM4, P2.3/PWM5, P2.4/PWMFLT]
1	PWM2/PWM3/PWM4/PWM5/PWMFLT在[P2.7/PWM2_2, P4.5/PWM3_2, P4.4/PWM4_2, P4.2/PWM5_2, P0.5/PWMFLT_2]

PWM6/PWM7可在2个地方切换，由 PWM67_S 控制位来选择	
PWM67_S	切换PWM6/PWM7管脚
0	PWM6/PWM7在[P1.6/PWM6, P1.7/PWM7]
1	PWM6/PWM7在[P0.7/PWM6_2, P0.6/PWM7_2]

SPI可在3个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI可在P1/P2/P4之间来回切换
0	0	SPI在[P1.2/SS,P1.3/MOSI,P1.4/MISO,P1.5/SCLK]
0	1	SPI在[P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI在[P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	无效

STC MCU Limited

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000

串口1/**S1**可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD,P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2,P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1] 串口1在P1口时要使用内部时钟
1	1	无效

串口2/**S2**可在2个地方切换，由 S2_S 控制位来选择	
S2_S	S2可在P1/P4之间来回切换
0	串口2/S2在[P1.0/RxD2,P1.1/TxD2]
1	串口2/S2在[P4.6/RxD2_2,P4.7/TxD2_2]

串口3/**S3**可在2个地方切换，由 S3_S 控制位来选择	
S3_S	S3可在P0/P5之间来回切换
0	串口3/S3在[P0.0/RxD3,P0.1/TxD3]
1	串口3/S3在[P5.0/RxD3_2,P5.1/TxD3_2]

串口4/**S4**可在2个地方切换，由 S4_S 控制位来选择	
S4_S	S4可在P0/P5之间来回切换
0	串口4/S4在[P0.2/RxD4,P0.3/TxD4]
1	串口4/S4在[P5.2/RxD4_2,P5.3/TxD4_2]

DPS: DPTR registers select bit. DPTR 寄存器选择位

0: DPTR0 is selected　　DPTR0被选择

1: DPTR1 is selected　　DPTR1被选择

STC MCU Limited

1.11.1 CCP/PWM/PCA在多个口之间切换的测试程序(C和汇编)

CCP：是下列英文单词的缩写
Capture(捕获),Compare(比较),PWM(脉宽调制)

1.C程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 CCP在多个口之间切换举例---------------------------------------------------*/
/* --- 研发顾问QQ：  800003751---------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

#define   FOSC    18432000L

//-----------------------------------------

sfr       P_SW1   =       0xA2;           //外设功能切换寄存器1

#define   CCP_S0  0x10                    //P_SW1.4
#define   CCP_S1  0x20                    //P_SW1.5

//-----------------------------------------
void main()
{
          ACC     =       P_SW1;
          ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=0
          P_SW1   =       ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1)

//        ACC     =       P_SW1;
//        ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=1 CCP_S1=0
//        ACC     |=      CCP_S0;                 //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2)
//        P_SW1   =       ACC;
//
//        ACC     =       P_SW1;
//        ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=1
//        ACC     |=      CCP_S1;                 //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3)
//        P_SW1   =       ACC;
          while (1);                              //程序终止
}
```

STC MCU Limited

2.汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 CCP在多个口之间切换举例----------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define   FOSC    18432000L

//-----------------------------------------
P_SW1   EQU     0A2H                    //外设功能切换寄存器1

CCP_S0  EQU     10H                     //P_SW1.4
CCP_S1  EQU     20H                     //P_SW1.5

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

//-----------------------------------------

        ORG     0100H

MAIN:
        MOV     SP,     #3FH

        MOV     A,      P_SW1
        ANL     A,      #0CFH           //CCP_S0=0 CCP_S1=0
        MOV     P_SW1,  A               //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1)

//      MOV     A,      P_SW1
//      ANL     A,      #0CFH           //CCP_S0=1 CCP_S1=0
//      ORL     A,      #CCP_S0         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2)
//      MOV     P_SW1,  A
//
//      MOV     A,      P_SW1
//      ANL     A,      #0CFH           //CCP_S0=0 CCP_S1=1
//      ORL     A,      #CCP_S1         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3)
//      MOV     P_SW1,  A
        SJMP    $                       //程序终止

        END
```

STC MCU Limited

1.11.2 PWM2/3/4/5/PWMFLT在多个口之间切换的测试程序(C和汇编)

1.C程序:

```
/*---------------------- PWM(脉宽调制)------------------------------------------------------*/
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 PWM2/3/4/5/PWMFLT在多个口之间切换举例----------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*-------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

#define   FOSC    18432000L

//-----------------------------------------

sfr       P_SW2   =       0xBA;           //外设功能切换寄存器2

#define   PWM2345_S       0x10            //P_SW2.4


//-----------------------------------------

void main()
{
          P_SW2   &=      ~PWM2345_S;     //PWM2345_S=0 ( P3.7/PWM2, P2.1/PWM3,
                                          //P2.2/PWM4, P2.3/PWM5, P2.4/PWMFLT )

//        P_SW2   |=      PWM2345_S;      //PWM2345_S=1 (P2.7/PWM2_2,  P4.5/PWM3_2,
                                          //P4.4/PWM4_2, P4.2/PWM5_2, P0.5/PWMFLT_2)

          while (1);                      //程序终止
}
```

STC MCU Limited

2.汇编程序：

```
/*---------------------- PWM(脉宽调制)------------------------------------------------------------*/
/*-------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 PWM2/3/4/5/PWMFLT在多个口之间切换举例----------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*-------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define   FOSC    18432000L

//-----------------------------------------

P_SW2           EQU     0BAH            //外设功能切换寄存器2

PWM2345_S       EQU     10H             //P_SW2.4

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

//-----------------------------------------

        ORG     0100H

MAIN:
        MOV     SP,     #3FH

        ANL     P_SW2,  #NOT    PWM2345_S       //PWM2345_S=0 ( P3.7/PWM2, P2.1/PWM3,
                                                //P2.2/PWM4, P2.3/PWM5, P2.4/PWMFLT )

//      ORL     P_SW2,  #PWM2345_S              //PWM2345_S=1 (P2.7/PWM2_2,  P4.5/PWM3_2,
                                                //P4.4/PWM4_2, P4.2/PWM5_2, P0.5/PWMFLT_2)

        SJMP    $                       //程序终止

        END
```

STC MCU Limited

1.11.3 PWM6/PWM7在多个口之间切换的测试程序(C和汇编)

1.C程序:

```
/*---------------------- PWM(脉宽调制)------------------------------------------------------*/
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 PWM6/PWM7在多个口之间切换举例---------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*-------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

#define   FOSC    18432000L

//-----------------------------------------

sfr       P_SW2   =       0xBA;               //外设功能切换寄存器2

#define   PWM67_S         0x20                //P_SW2.5


//-----------------------------------------

void main()
{
          P_SW2   &=      ~PWM67_S;           //PWM67_S=0 ( P1.6/PWM6, P1.7/PWM7 )

//        P_SW2   |=      PWM67_S;            //PWM67_S=1 ( P0.7/PWM6_2, P0.6/PWM7_2 )

          while (1);                          //程序终止
}
```

STC MCU Limited

2.汇编程序：

```
/*---------------------- PWM(脉宽调制)------------------------------------------------------*/
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 PWM6/PWM7在多个口之间切换举例---------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*-------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define   FOSC    18432000L

//-----------------------------------------

P_SW2           EQU     0BAH                    //外设功能切换寄存器2

PWM67_S         EQU     20H                     //P_SW2.5

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                            //复位入口

//-----------------------------------------

        ORG     0100H

MAIN:
        MOV     SP,     #3FH

        ANL     P_SW2,  #NOT    PWM67_S         //PWM67_S=0 ( P1.6/PWM6, P1.7/PWM7 )

//      ORL     P_SW2,  #PWM67_S                //PWM67_S=1 ( P0.7/PWM6_2, P0.6/PWM7_2 )

        SJMP    $                               //程序终止

        END
```

STC MCU Limited

1.11.4 SPI在多个口之间切换的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 SPI在多个口之间切换举例----------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

#define FOSC 18432000L

//-----------------------------------------

sfr     P_SW1   =       0xA2;                   //外设功能切换寄存器1

#define SPI_S0  0x04                            //P_SW1.2
#define SPI_S1  0x08                            //P_SW1.3

//-----------------------------------------

void main()
{
        ACC     =       P_SW1;
        ACC     &=      ~(SPI_S0 | SPI_S1);     //SPI_S0=0 SPI_S1=0
        P_SW1   =       ACC;                    //(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)

//      ACC     =       P_SW1;
//      ACC     &=      ~(SPI_S0 | SPI_S1);     //SPI_S0=1 SPI_S1=0
//      ACC     |=      SPI_S0;         //(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
//      P_SW1   =       ACC;
//
//      ACC     =       P_SW1;
//      ACC     &=      ~(SPI_S0 | SPI_S1);     //SPI_S0=0 SPI_S1=1
//      ACC     |=      SPI_S1;         //(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
//      P_SW1   =       ACC;

        while (1);                              //程序终止
}
```

STC MCU Limited

2.汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 SPI在多个口之间切换举例----------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define   FOSC    18432000L

//-----------------------------------------

P_SW1   EQU     0A2H                    //外设功能切换寄存器1
SPI_S0  EQU     04H                     //P_SW1.2
SPI_S1  EQU     08H                     //P_SW1.3

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

//-----------------------------------------

        ORG     0100H

MAIN:
        MOV     SP,     #3FH

        MOV     A,      P_SW1
        ANL     A,      #0F3H           //SPI_S0=0 SPI_S1=0
        MOV     P_SW1,  A               //(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)

//      MOV     A,      P_SW1
//      ANL     A,      #0F3H           //SPI_S0=1 SPI_S1=0
//      ORL     A,      #SPI_S0         //(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
//      MOV     P_SW1,  A
//
//      MOV     A,      P_SW1
//      ANL     A,      #0F3H           //SPI_S0=0 SPI_S1=1
//      ORL     A,      #SPI_S1         //(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
//      MOV     P_SW1,  A

        SJMP    $                       //程序终止

        END
```

STC MCU Limited

1.11.5 串口1在多个口之间切换的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列  串行口1在多个口之间切换举例----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

#define FOSC 18432000L

//-----------------------------------------

sfr     P_SW1   =       0xA2;                   //外设功能切换寄存器1
#define S1_S0   0x40                            //P_SW1.6
#define S1_S1   0x80                            //P_SW1.7

//-----------------------------------------

void main()
{
        ACC     =       P_SW1;
        ACC     &=      ~(S1_S0 | S1_S1);       //S1_S0=0 S1_S1=0
        P_SW1   =       ACC;                    //(P3.0/RxD, P3.1/TxD)

//      ACC     =       P_SW1;
//      ACC     &=      ~(S1_S0 | S1_S1);       //S1_S0=1 S1_S1=0
//      ACC     |=      S1_S0;                  //(P3.6/RxD_2, P3.7/TxD_2)
//      P_SW1   =       ACC;
//
//      ACC     =       P_SW1;
//      ACC     &=      ~(S1_S0 | S1_S1);       //S1_S0=0 S1_S1=1
//      ACC     |=      S1_S1;                  //(P1.6/RxD_3, P1.7/TxD_3)
//      P_SW1   =       ACC;

        while (1);                              //程序终止
}
```

STC MCU Limited

2.汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 串行口1在多个口之间切换举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define FOSC 18432000L

//-----------------------------------------
P_SW1   EQU     0A2H                            //外设功能切换寄存器1

S1_S0   EQU     40H             //P_SW1.6
S1_S1   EQU     80H             //P_SW1.7

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                            //复位入口

//-----------------------------------------

        ORG     0100H

MAIN:
        MOV     SP,     #3FH

        MOV     A,      P_SW1
        ANL     A,      #03FH                   //S1_S0=0 S1_S1=0
        MOV     P_SW1,  A                       //(P3.0/RxD, P3.1/TxD)

//      MOV     A,      P_SW1
//      ANL     A,      #03FH                   //S1_S0=1 S1_S1=0
//      ORL     A,      #S1_S0                  //(P3.6/RxD_2, P3.7/TxD_2)
//      MOV     P_SW1,  A
//
//      MOV     A,      P_SW1
//      ANL     A,      #03FH                   //S1_S0=0 S1_S1=1
//      ORL     A,      #S1_S1                  //(P1.6/RxD_3, P1.7/TxD_3)
//      MOV     P_SW1,  A

        SJMP    $                               //程序终止

        END
```

STC MCU Limited

1.11.6 串口2在多个口之间切换的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列  串行口2在多个口之间切换举例----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

#define FOSC 18432000L

//-----------------------------------------

sfr     P_SW2   =       0xBA;                   //外设功能切换寄存器2
#define S2_S    0x01                            //P_SW2.0

//-----------------------------------------

void main()
{
        P_SW2   &=      ~S2_S;                  //S2_S0=0 (P1.0/RxD2, P1.1/TxD2)

//      P_SW2   |=      S2_S;           `       //S2_S0=1 (P4.6/RxD2_2, P4.7/TxD2_2)

        while (1);                              //程序终止
}
```

STC MCU Limited

2.汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 串行口2在多个口之间切换举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define   FOSC    18432000L

//-----------------------------------------

P_SW2   EQU     0BAH                    //外设功能切换寄存器2

S2_S    EQU     01H                     //P_SW2.0

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

//-----------------------------------------

        ORG     0100H

MAIN:
        MOV     SP,     #3FH

        ANL     P_SW2,  #NOT S2_S       //S2_S0=0 (P1.0/RxD2, P1.1/TxD2)

//      ORL     P_SW2,  #S2_S           //S2_S0=1 (P4.6/RxD2_2, P4.7/TxD2_2)

        SJMP    $                       //程序终止

        END
```

STC MCU Limited

1.11.7 串口3在多个口之间切换的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 串口3在多个口之间切换举例-----------------------*/
/* --- 研发顾问QQ：800003751----------------- ----------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序 */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序 */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

#define    FOSC    18432000L

//-----------------------------------------

sfr        P_SW2   =       0xBA;           //外设功能切换寄存器2

#define    S3_S            0x02            //P_SW2.1

//-----------------------------------------

void main()
{
           P_SW2   &=      ~S3_S;          //S3_S0=0 (P0.0/RxD3, P0.1/TxD3)

//         P_SW2   |=      S3_S;           //S3_S0=1 (P5.0/RxD3_2, P5.1/TxD3_2)

           while (1);                      //程序终止
}
```

STC MCU Limited

2.汇编程序：

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 串口3在多个口之间切换举例-----------------------*/
/* --- 研发顾问QQ：800003751---------------- ----------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序 */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序 */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define   FOSC    18432000L

//-----------------------------------------

P_SW2   EQU     0BAH                    //外设功能切换寄存器2

S3_S    EQU     02H                     //P_SW2.1

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

//-----------------------------------------

        ORG     0100H

MAIN:
        MOV     SP,     #3FH

        ANL     P_SW2,  #NOT S3_S       //S3_S0=0 (P0.0/RxD3, P0.1/TxD3)

//      ORL     P_SW2,  #S3_S           //S3_S0=1 (P5.0/RxD3_2, P5.1/TxD3_2)

        SJMP    $                       //程序终止

        END
```

STC MCU Limited

1.11.8 串口4在多个口之间切换的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 串口4在多个口之间切换举例-----------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序 */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序 */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可*/
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

#define    FOSC    18432000L

//-----------------------------------------

sfr        P_SW2   =       0xBA;           //外设功能切换寄存器2

#define    S4_S    0x04                    //P_SW2.2

//-----------------------------------------

void main()
{
           P_SW2   &=      ~S4_S;           //S4_S0=0 (P0.2/RxD4, P0.3/TxD4)

//         P_SW2   |=      S4_S;            //S4_S0=1 (P5.2/RxD4_2, P5.3/TxD4_2)

           while (1);                       //程序终止
}
```

STC MCU Limited

2.汇编程序：

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 串口4在多个口之间切换举例----------------------*/
/* --- 研发顾问QQ：800003751---------------- ----------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序 */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序 */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define   FOSC    18432000L

//-----------------------------------------

P_SW2   EQU     0BAH                    //外设功能切换寄存器2

S4_S0   EQU     04H                     //P_SW2.2

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

//-----------------------------------------

        ORG     0100H

MAIN:
        MOV     SP,     #3FH

        ANL     P_SW2,  #NOT  S4_S      //S4_S0=0 (P0.2/RxD4, P0.3/TxD4)

//      ORL     P_SW2,  #S4_S           //S4_S0=1 (P5.2/RxD4_2, P5.3/TxD4_2)

        SJMP    $                       //程序终止

        END
```

STC MCU Limited

1.12 每个单片机具有全球唯一身份证号码(ID号)及其测试程序

STC最新一代STC15系列单片机出厂时都具有全球唯一身份证号码(ID号)。最新STC15系列单片机的程序存储器的最后7个字节单元的值是全球唯一ID号，用户不可修改，但IAP15系列单片机的整个程序区是开放的，可以修改。建议利用全球唯一ID号加密时，使用STC15系列单片机，并将EEPROM功能使用上，从EEPROM起始地址0000H开始使用，可以有效杜绝对全球唯一ID号的攻击。

除程序存储器的最后7个字节单元的内容是全球唯一ID号外，单片机内部RAM的F1H ~ F7H单元 (对于STC15F101W系列及STC15W104SW系列单片机是内部RAM的71H - 77H单元)的内容也为全球唯一ID号。用户可以在单片机上电后读取内部RAM单元F1H - F7H (对于STC15F101W系列及STC15W104SW系列单片机是内部RAM单元71H - 77H)连续7个单元的值来获取此单片机的唯一身份证号码(ID号), 使用“ MOV @Ri” 指令来读取。如果用户需要用全球唯一ID号进行用户自己的软件加密，建议用户在程序的多个地方有技巧地判断自己的用户程序有无被非法修改，提高解密的难度，防止解密者修改程序，绕过对全球唯一ID号的判断。

使用程序区的最后7个字节的全球唯一ID号比使用内部RAM单元 F1H - F7H (或内部RAM单元71H - 77H)的全球唯一ID号更难被攻击。建议用户使用程序区最后7个字节的全球唯一ID号，而不要使用内部RAM单元 F1H - F7H (或内部RAM单元71H - 77H)的全球唯一ID号。

//从RAM区和程序区获取全球唯一身份证号码(ID号)的程序举例

1. C程序

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 获取全球唯一身份证号码(ID号)举例----------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*//
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;

#define    URMD   0                 //0:使用定时器2作为波特率发生器
                                    //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                    //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
```

```
sfr     T2H     =       0xd6;                   //定时器2高8位
sfr     T2L     =       0xd7;                   //定时器2低8位

sfr     AUXR    =       0x8e;                   //辅助寄存器

#define ID_ADDR_RAM             0xf1    //ID号的存放在RAM区的地址为0F1H

//ID号的存放在程序区的地址为程序空间的最后7字节
//#define ID_ADDR_ROM   0x03f9          //1K程序空间的MCU(如STC15F201EA, STC15F101EA)
//#define ID_ADDR_ROM   0x07f9          //2K程序空间的MCU(如STC15F402AD,
                                        //STC15F202EA, STC15F102EA)
//#define ID_ADDR_ROM   0x0bf9          //3K程序空间的MCU(如STC15F203EA, STC15F103EA)
//#define ID_ADDR_ROM   0x0ff9          //4K程序空间的MCU(如STC15F404AD, STC15F204EA,
                                        //STC15F104EA)
//#define ID_ADDR_ROM   0x13f9          //5K程序空间的MCU(如 STC15F206EA, STC15F106EA)
//#define ID_ADDR_ROM   0x1ff9          //8K程序空间的MCU(如STC15F2K08S2,  STC15F1K08AD,
                                        //STC15F408AD)
//#define ID_ADDR_ROM   0x27f9          //10K程序空间的MCU(如STC15F410AD)
//#define ID_ADDR_ROM   0x2ff9          //12K程序空间的MCU(如STC15F408AD)
//#define ID_ADDR_ROM   0x3ff9          //16K程序空间的MCU(如STC15F2K16S2,
                                        //STC15F1K16AD)
//#define ID_ADDR_ROM   0x4ff9          //20K程序空间的MCU(如STC15F2K20S2)
//#define ID_ADDR_ROM   0x5ff9          //24K程序空间的MCU
//#define ID_ADDR_ROM   0x6ff9          //28K程序空间的MCU
//#define ID_ADDR_ROM   0x7ff9          //32K程序空间的MCU(如STC15F2K32S2)
//#define ID_ADDR_ROM   0x9ff9          //40K程序空间的MCU(如STC15F2K40S2)
//#define ID_ADDR_ROM   0xbff9          //48K程序空间的MCU(如STC15F2K48S2)
//#define ID_ADDR_ROM   0xcff9          //52K程序空间的MCU(如STC15F2K52S2)
//#define ID_ADDR_ROM   0xdff9          //56K程序空间的MCU(如STC15F2K56S2)
#define ID_ADDR_ROM     0xeff9          //60K程序空间的MCU(如STC15W4K60S4)

//-----------------------------------------

void   InitUart();
void   SendUart(BYTE dat);

//-----------------------------------------
void   main()
{
        BYTE    idata   *iptr;
        BYTE    code    *cptr;
        BYTE    i;

        InitUart();                             //串口初始化
```

STC MCU Limited

```
	iptr =  ID_ADDR_RAM;                    //从RAM区读取ID号
	for (i=0; i<7; i++)                     //读7个字节
	{
		SendUart(*iptr++);              //发送ID到串口
	}
	cptr = ID_ADDR_ROM;                     //从程序区读取ID号
	for (i=0; i<7; i++)                     //读7个字节
	{
		SendUart(*cptr++);              //发送ID到串口
	}

	while (1);                              //程序终止
}
/*---------------------------
串口初始化
---------------------------*/
void InitUart()
{
	SCON	=	0x5a;                   //设置串口为8位可变波特率
#if	URMD	==	0
	T2L	=	0xd8;                   //设置波特率重装值
	T2H	=	0xff;                   //115200 bps(65536-18432000/4/115200)
	AUXR	=	0x14;                   //T2为1T模式, 并启动定时器2
	AUXR	|=	0x01;                   //选择定时器2为串口1的波特率发生器
#elif	URMD	==	1
	AUXR	=	0x40;                   //定时器1为1T模式
	TMOD	=	0x00;                   //定时器1为模式0(16位自动重载)
	TL1	=	0xd8;                   //设置波特率重装值
	TH1	=	0xff;                   //115200 bps(65536-18432000/4/115200)
	TR1	=	1;                      //定时器1开始启动
#else
	TMOD	=	0x20;                   //设置定时器1为8位自动重装载模式
	AUXR	=	0x40;                   //定时器1为1T模式
	TH1	=	TL1	= 0xfb;         //115200 bps(256 - 18432000/32/115200)
	TR1	=	1;
#endif
}

/*---------------------------
发送串口数据
---------------------------*/
void SendUart(BYTE dat)
{
	while (!TI);                            //等待前面的数据发送完成
	TI = 0;                                 //清除发送完成标志
	SBUF = dat;                             //发送串口数据
}
```

STC MCU Limited

2. 汇编程序

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 获取全球唯一身份证号码(ID号)举例-----------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define   URMD   0                  //0:使用定时器2作为波特率发生器
                                    //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                    //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
T2H       DATA   0D6H               //定时器2高8位
T2L       DATA   0D7H               //定时器2低8位
AUXR      DATA   08EH               //辅助寄存器
//-----------------------------------------

#define   ID_ADDR_RAM   0xf1        //ID号的存放在RAM区的地址为0F1H

                                    //ID号的存放在程序区的地址为程序空间的最后7字节
//#define ID_ADDR_ROM   0x03f9      //1K程序空间的MCU(如STC15F201EA, STC15F101EA)
//#define ID_ADDR_ROM   0x07f9      //2K程序空间的MCU(如 STC15F402AD, STC15F202EA,
                                    // STC15F102EA)
//#define ID_ADDR_ROM   0x0bf9      //3K程序空间的MCU如STC15F203EA, STC15F103EA)
//#define ID_ADDR_ROM   0x0ff9      //4K程序空间的MCU(如STC15F404AD, STC15F204EA,
                                    //STC15F104EA)
//#define ID_ADDR_ROM   0x13f9      //5K程序空间的MCU(如STC15F206EA, STC15F106EA)
//#define ID_ADDR_ROM   0x1ff9      //8K程序空间的MCU(如STC15F2K08S2,  STC15F1K08AD,
                                    //STC15F408AD)
//#define ID_ADDR_ROM   0x27f9      //10K程序空间的MCU(如 STC15F410AD)
//#define ID_ADDR_ROM   0x2ff9      //12K程序空间的MCU(如STC15F408AD)
//#define ID_ADDR_ROM   0x3ff9      //16K程序空间的MCU(如STC15F2K16S2)
//#define ID_ADDR_ROM   0x4ff9      //20K程序空间的MCU(如STC15F2K20S2)
//#define ID_ADDR_ROM   0x5ff9      //24K程序空间的MCU
//#define ID_ADDR_ROM   0x6ff9      //28K程序空间的MCU
//#define ID_ADDR_ROM   0x7ff9      //32K程序空间的MCU(如STC15F2K32S2)
//#define ID_ADDR_ROM   0x9ff9      //40K程序空间的MCU(如STC15F2K40S2)
//#define ID_ADDR_ROM   0xbff9      //48K程序空间的MCU(如STC15F2K48S2)
//#define ID_ADDR_ROM   0xcff9      //52K程序空间的MCU(如STC15F2K52S2)
```

```
//#define  ID_ADDR_ROM   0xdff9          //56K程序空间的MCU(如STC15F2K56S2)
#define   ID_ADDR_ROM   0xeff9          //60K程序空间的MCU(如STC15W4K60S4)

//-----------------------------------------

//-----------------------------------------
          ORG     0000H
          LJMP    MAIN                          //复位入口

//-----------------------------------------

          ORG     0100H

MAIN:
          MOV     SP,       #3FH

          LCALL   INIT_UART                     //串口初始化

          MOV   R0,         #ID_ADDR_RAM        //从RAM区读取ID号
          MOV   R1,         #7                  //读7个字节
NEXT1:
          MOV   A,          @R0
          LCALL SEND_UART                       //发送ID到串口
          INC   R0
          DJNZ  R1,         NEXT1

          MOV   DPTR,       #ID_ADDR_ROM        //从程序区读取ID号
          MOV   R1,         #7                  //读7个字节
NEXT2:
          CLR     A
          MOVC    A,        @A+DPTR
          LCALL   SEND_UART                     //发送ID到串口
          INC     DPTR
          DJNZ    R1,       NEXT2

          SJMP    $                             //程序终止

/*----------------------------
串口初始化
----------------------------*/
INIT_UART:
          MOV     SCON,     #5AH                //设置串口为8位可变波特率
#if       URMD == 0
          MOV     T2L,      #0D8H               //设置波特率重装值(65536-18432000/4/115200)
          MOV     T2H,      #0FFH
          MOV     AUXR,     #14H                //T2为1T模式, 并启动定时器2
          ORL     AUXR,     #01H                //选择定时器2为串口1的波特率发生器
```

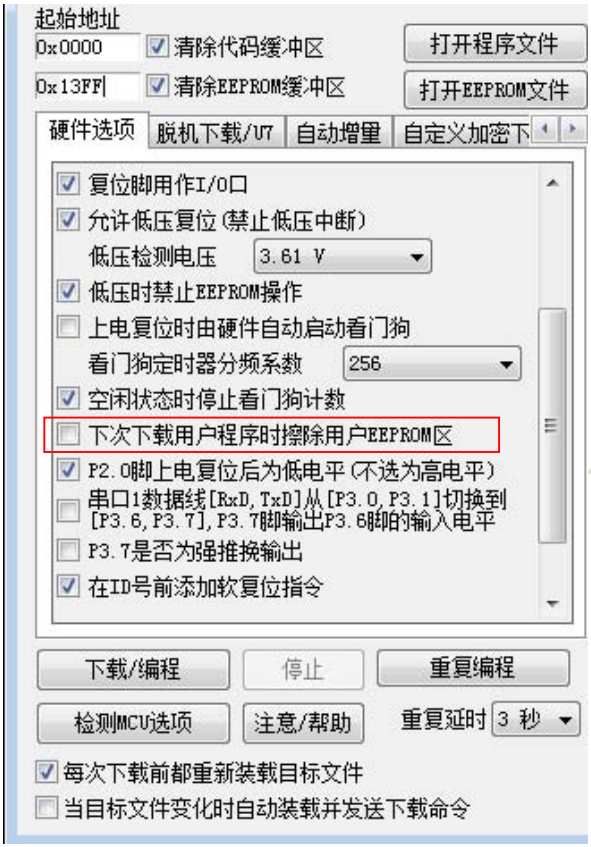
STC MCU Limited

```
#elif   URMD == 1
        MOV     AUXR,   #40H            //定时器1为1T模式
        MOV     TMOD,   #00H            //定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H           //设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                     //定时器1开始运行
#else
        MOV     TMOD,   #20H            //设置定时器1为8位自动重装载模式
        MOV     AUXR,   #40H            //定时器1为1T模式
        MOV     TL1,    #0FBH           //115200 bps(256 - 18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1
#endif
        RET

/*---------------------------
发送串口数据
入口参数: ACC
出口参数: 无
---------------------------*/
SEND_UART:
        JNB     TI,     $               //等待前面的数据发送完成
        CLR     TI                      //清除发送完成标志
        MOV     SBUF,   A               //发送串口数据
        RET

        END
```

STC MCU Limited

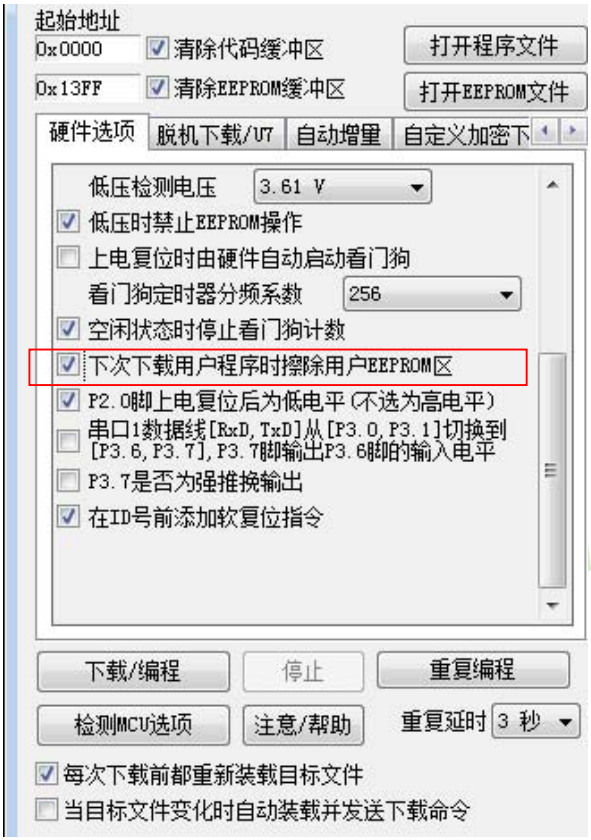
1.13 关于ID号在大批量生产中的应用方法(较多用户的用法)

（1）先烧一个程序进去(选择下次下载用户程序时不擦除用户EEPROM区)；



（2）读程序区的ID号(STC15系列是程序区的最后7个字节)，经用户自己的复杂的加密算法对程序区的ID号加密运算后生成一个新的数——用户自加密ID号，写入STC15系列用户EEPROM区的EEPROM；

（3）再烧一个最终出厂的程序进去(选择下次下载用户程序时将用户EEPROM区一并擦除)，如下图所示；

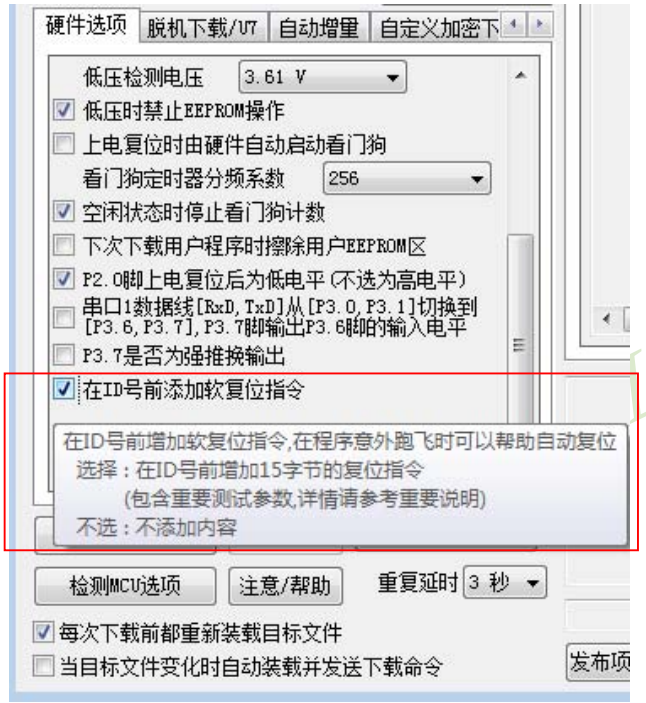


（4）在用户程序区的多处读程序区的ID号和用户自加密ID号比较(经用户自己的复杂的解密算法解密后),如不对应,则6个月后随机异常,或200次开机后随机异常最终出厂的程序不含加密算法

（5）另外,在程序区的多个地方判断用户自己的程序是否被修改,如被修改,则6个月后随机异常,或200次开机后随机异常,将不用的用户程序区用所谓的有效程序全部填满

1.14 在全球唯一ID号前添加软复位指令及重要测试参数

用户可以在STC-ISP下载编程工具中设置在全球唯一身份证号码(ID号)前增加15字节的软复位指令(包含重要测试参数)，具体设置方法见下图。这15字节的软复位指令可以在程序意外跑飞时帮助CPU自动复位。



该15字节的复位指令中包含一些重要的测试参数。当全球唯一ID号在程序存储器(ROM)的最后7个字节单元时，这些测试参数在ROM区的信息如下：

(1) 内部BandGap电压值(毫伏,高字节在前)(2字节，**程序空间最后第**8字节和第9字节)

例如：　STC15F104W　4K程序空间　　地址为0FF7H-0FF8H

STC15F2K60S2　60K程序空间　　地址为EFF7H-EFF8H

(2) 32K掉电唤醒定时器频率(高字节在前)(2字节，**程序空间最后第**10字节和第11字节)

例如：　STC15F104W　4K程序空间　　地址为0FF5H-0FF6H

STC15F2K60S2　60K程序空间　　地址为EFF5H-EFF6H

(3) 12MHz内部IRC设定参数值(3字节，**程序空间最后第**12字节、第13字节和第14字节)

例如：　STC15F104W　4K程序空间　　地址为0FF2H-0FF4H

STC15F2K60S2　60K程序空间　　地址为EFF2H-EFF4H

(4) 24MHz内部IRC设定参数值(3字节，**程序空间最后第**15字节、第16字节和第17字节)

例如：　STC15F104W　4K程序空间　　地址为0FEEH-0FF1H

STC15F2K60S2　60K程序空间　　地址为EFEEH-EFF1H

当全球唯一ID号在内部RAM单元时，这些测试参数在RAM区的信息如下：

(1) 24MHz内部IRC设定参数值1(1字节)

例如： STC15F104W 128字节RAM 地址为07FH-07FH

STC15F2K60S2 256以上字节RAM 地址为0FFH-0FFH

(2) 12MHz内部IRC设定参数值1(1字节)

例如： STC15F104W 128字节RAM 地址为07EH-07EH

STC15F2K60S2 256以上字节RAM 地址为0FEH-0FEH

(3) 24MHz内部IRC设定参数值2(1字节)

例如： STC15F104W 128字节RAM 地址为07DH-07DH

STC15F2K60S2 256以上字节RAM 地址为0FDH-0FDH

(4) 12MHz内部IRC设定参数值2(1字节)

例如： STC15F104W 128字节RAM 地址为07CH-07CH

STC15F2K60S2 256以上字节RAM 地址为0FCH-0FCH

(5) 24MHz内部IRC设定参数值3(1字节)

例如： STC15F104W 128字节RAM 地址为07BH-07BH

STC15F2K60S2 256以上字节RAM 地址为0FBH-0FBH

(6) 12MHz内部IRC设定参数值3(1字节)

例如： STC15F104W 128字节RAM 地址为07AH-07AH

STC15F2K60S2 256以上字节RAM 地址为0FAH-0FAH

(7) 32K掉电唤醒定时器频率(高字节在前)(2字节)

例如： STC15F104W 128字节RAM 地址为078H-079H

STC15F2K60S2 256以上字节RAM 地址为0F8H-0F9H

(8) 内部BandGap电压值(毫伏,高字节在前)(2字节)

例如： STC15F104W 128字节RAM 地址为06FH-070H

STC15F2K60S2 256以上字节RAM 地址为0EFH-0F0H

STC MCU Limited

1.15 现供货的STC15系列中未实现的计划功能

计划功能是指芯片原规格说明书中计划设计的功能。这些功能在单片机指南中已作介绍，但是由于某些原因，在芯片定型生产时，现供货的STC15F2K60S2系列C版本、STC15F408AD系列C版本及STC15W201S系列A版本的部分计划功能实际未设计进去，在此特别指出这些系列中未实现的计划功能，敬请广大客户留意！

1.15.1 现供货的STC15F2K60S2系列C版本中未实现的计划功能

1.15.1.1 现供货的STC15F2K60S2系列C版本的主时钟只可对外输出内部R/C时钟——将在STC15W2K60S2系列中修正

STC15F2K60S2系列芯片已经大批量供货一年了，现STC15F2K60S2系列已经发展到了C版本。STC15F2K60S2系列单片机的主时钟在P5.4/MCLKO口对外输出时钟，并可如下分频MCLK/1, MCLK/2, MCLK/4 (MCLK为主时钟频率)。现供货的STC15F2K60S2系列C版本单片机的主时钟在P5.4/MCLKO口只可以对外输出内部R/C时钟，暂时不可以对外输出外部输入的时钟或外部晶体振荡产生的时钟，特此说明，敬请广大客户留意！用户可通过在P1.6/XTAL2脚串一电阻将外部输入的时钟或外部晶体振荡产生的时钟进行对外输出。

STC15F2K60S2及STC15L2K60S2系列下一升级版本——STC15W2K60S2系列单片机将会设计实现该计划功能，其主时钟将既可以对外输出内部R/C时钟，也可对外输出外部输入的时钟或外部晶体振荡产生的时钟。同时，STC15W2K60S2系列单片机还将增加比较器的功能。

1.15.1.2 现供货的STC15F2K60S2系列C版本的管脚RxD/RxD2不能唤醒掉电/停机模式——将在STC15W2K60S2系列中修正

STC15F2K60S2系列芯片已经大批量供货一年了，现STC15F2K60S2系列已经发展到了C版本。现供货的STC15F2K60S2系列C版本单片机的RxD管脚和RxD2管脚暂时不能将掉电模式/停机模式唤醒，特此说明，敬请广大客户留意！但是，当RxD管脚切换至管脚P3.0/$\overline{\text{INT4}}$或P3.6/$\overline{\text{INT2}}$时，其与$\overline{\text{INT4}}$（外部中断4）或$\overline{\text{INT2}}$（外部中断2）占用同一管脚资源，若相应的$\overline{\text{INT4}}$（外部中断4）或$\overline{\text{INT2}}$（外部中断2）被允许，则该管脚资源可通过$\overline{\text{INT4}}$或$\overline{\text{INT2}}$来唤醒掉电模式/停机模式。

STC15F2K60S2及STC15L2K60S2系列下一升级版本——STC15W2K60S2系列单片机将会设计实现该计划功能，STC15W2K60S2系列单片机的RxD管脚和RxD2管脚将均可用于唤醒掉电模式/停机模式。同时，STC15W2K60S2系列单片机还将增加比较器的功能。

1.15.2 现供货的STC15F408AD系列C版本中未实现的计划功能

1.15.2.1 现供货的STC15F408AD系列C版本的RxD管脚不能唤醒掉电模式/停机模式——将在STC15W408AS系列中修正

STC15F408AD系列芯片已经开始供货了。现供货的STC15F408AD系列C版本单片机的RxD管脚暂时不能将掉电模式/停机模式唤醒，特此说明，敬请广大客户留意！但是，RxD管脚与$\overline{\text{INT4}}$（外部中断4）占用同一管脚资源，若$\overline{\text{INT4}}$（外部中断4）被允许，则该管脚资源可通过$\overline{\text{INT4}}$来唤醒掉电模式/停机模式。

STC15F408AD及STC15L408AD系列下一升级版本——STC15W408AS系列单片机将会设计实现该计划功能，STC15W408AS系列单片机的RxD管脚将可用于唤醒掉电模式/停机模式。

1.16 部分15系列单片机的特别注意事项

1.16.1 SPI的特别注意事项(仅针对以15F和15L开头的单片机)——只支持SPI主机模式，不支持SPI从机模式

STC单片机中以15F和15L开头且有SPI功能的单片机(如STC15F2K60S2型号及STC15L408AD单片机)的SPI从机模式暂不能使用，但它们的SPI主机模式可正常使用。因此，建议用户不要使用以15F和15L开头且有SPI功能的单片机的SPI从机模式。

注意，以15W开头的单片机不存在上述问题，以15W开头且有SPI功能的单片机既支持SPI主机模式，也支持SPI从机模式。如，STC15W408S、STC15W1K16S等型号单片机既支持SPI主机模式，也支持SPI从机模式

1.16.2 进入掉电唤醒模式的特别注意事项(仅针对以15L开头的单片机)——以15L开头的单片机进入掉电模式前必须启动掉电唤醒定时器

STC单片机中以15L开头的单片机(如STC15L2K60S2型号单片机)如需进入"掉电模式",则它进入"掉电模式"前必须启动掉电唤醒定时器（功耗为3uA）,且其掉电唤醒定时器不超过3秒（约2秒)要唤醒一次。而以15F和15W开头的单片机则不需要。如，STC15F2K60S2、STC15W408S等型号单片机在进入“掉电模式”前不需要启动掉电唤醒定时器。

STC MCU Limited

1.16.3 STC15W201S系列A版本单片机的比较器下降沿中断不响应

——将在STC15W201S系列B版本中修正

STC15W201S系列的A版本单片机正大批量现货供应中，STC15W201S系列包括STC15W201S型号、STC15W202S型号、STC15W203S型号、STC15W204S型号、IAP15W205S型号及IRC15W207S型号单片机。当仅允许该系列单片机的比较器下降沿中断时，该比较器的下降沿中断暂不能使用。

但是，STC15W201S系列A版本单片机的比较器下降沿中断不是绝对不能使用，用户可以通过以下两种办法解决这一问题：一、STC15W201S系列A版本的比较器上升沿中断是可正常使用的，且当用户将其比较器上升沿中断和下降沿中断都允许后，该比较器上升沿中断和下降沿中断都可以正常使用。由于比较器比较结果标志位CMPRES(CMPCR1.0)是正确的，因此在比较器中断服务程序中查询比较器比较结果标志位CMPRES(CMPCR1.0)的值可判断单片机进入的是比较器上升沿中断还是比较器下降沿中断，如果CMPRES/CMPCR1.0=1，即CMP+的电平高于CMP-的电平(或内部BandGap参考电压的电平)，则表示单片机进入的是比较器上升沿中断；反之，如果CMPRES(CMPCR1.0)=0，即CMP+的电平低于CMP-的电平(或内部BandGap参考电压的电平)，则表示单片机进入的是比较器下降沿中断，此时比较器下降沿中断是可正常使用的，这是解决办法之一；二、STC15W201S系列A版本的比较器比较结果标志位(CMPRES)是正确的，因此用户还可用软件查询方式解决该这一问题。

对于上述问题，我们将在STC15W201S系列的下一版本，即STC15W201S系列B版本中修正。

1.16.4 STC15W408S及STC15W1K16S系列T0CLKO时钟输出功能的注意事

——若要使用T0CLKO时钟输出功能，须将P3.5口设置为强推挽输出

STC15W408S系列单片机和STC15W1K16S系列单片机均已开始供货，但STC15W408S系列单片机和STC15W1K16S系列单片机若要使用其T0CLKO时钟输出功能，必须将P3.5/T0CLKO口设置为强推挽输出模式。只有将P3.5口设置成了强推挽输出模式，STC15W408S系列单片机和STC15W1K16S系列单片机的T0CLKO时钟输出功能才有效。

第2章 STC15系列的时钟、复位及省电模式

2.1 STC15系列单片机的时钟

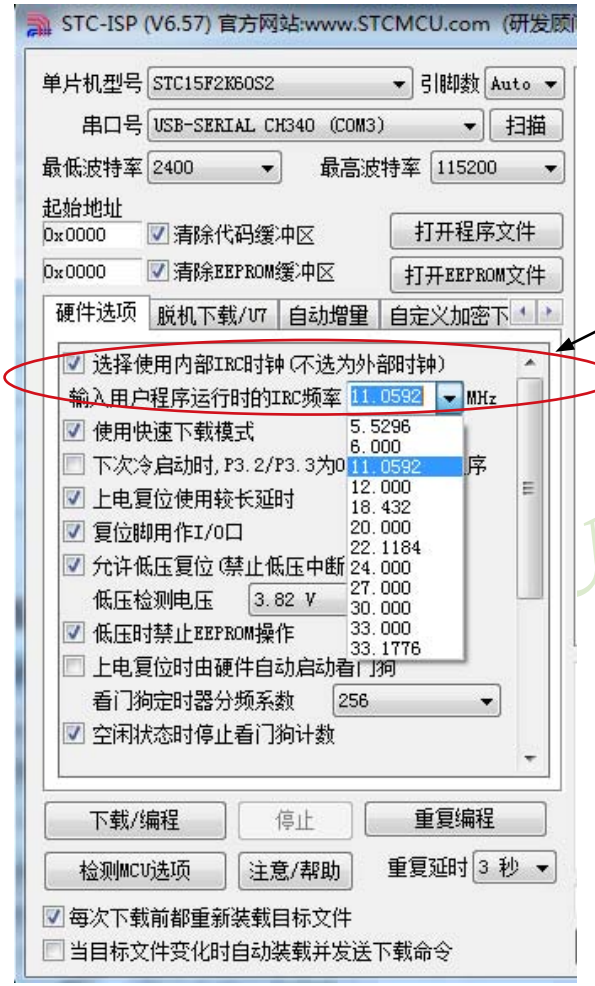
STC15F2K60S2系列、STC15W4K60S4系列、STC15W408AS系列和STC15F408AD系列单片机有两个时钟源：内部高精度R/C时钟和外部时钟(外部输入的时钟或外部晶体振荡产生的时钟)。而STC15F101W系列、STC15W201S系列、STC15W408S系列和STC15W1K16S系列无外部时钟只有内部高精度R/C时钟。内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃), 常温下温飘±0.6%(-20℃~+65℃)

STC15系列单片机的的时钟源见下表所示。

时钟源类型 / 单片机型号	内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)	外部时钟（外部输入的时钟或外部晶体振荡产生的时钟）
STC15F101W系列	√	
STC15F408AD系列	√	√
STC15W201S系列	√	
STC15W408AS系列	√	√
STC15W408S系列	√	
STC15W1K16S系列	√	
STC15F2K60S2系列	√	√
STC15W4K60S4系列	√	√

上表中√表示对应的系列有相应的时钟源。

2.1.1 STC15系列单片机的内部可配置时钟



选择内部高精度R/C时钟(IRC)的频率
(用户也可以直接输入)

2.1.2 主时钟分频和分频寄存器

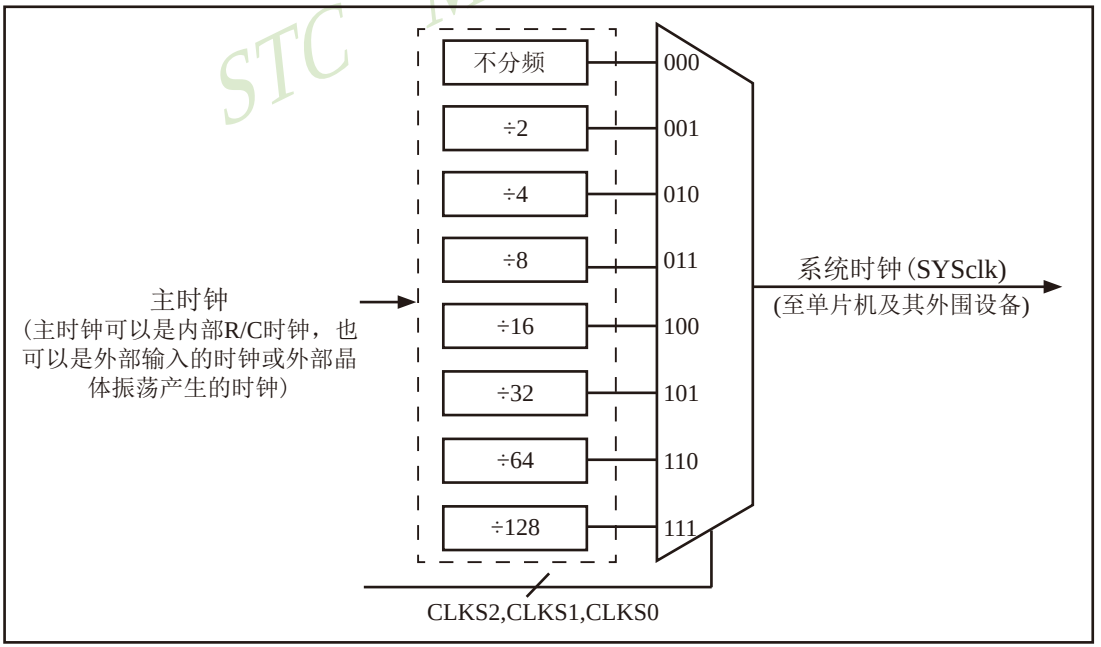
如果希望降低系统功耗，可对时钟进行分频。利用时钟分频控制寄存器CLK_DIV(PCON2)可进行时钟分频，从而使单片机在较低频率下工作。

时钟分频寄存器CLK_DIV (PCON2)各位的定义如下：

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D转换的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。



时钟结构

时钟分频寄存器CLK_DIV (PCON2)其他各位的说明如下：

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， **输出时钟频率** = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， **输出时钟频率** = MCLK / 4

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK是指主时钟频率。

STC15W4K60S4系列单片机在MCLKO/P5.4口或MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列等)均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15W系列单片机通过CLK_DIV.3/MCLKO_2位来选择是在MCLKO/P5.4口对外输出主时钟，还是在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位

0：在MCLKO/P5.4口对外输出主时钟；

1：在MCLKO_2/XTAL2/P1.6口对外输出主时钟；

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

ADRJ：ADC转换结果调整

0：ADC_RES[7:0]存放高8位ADC结果，ADC_RESL[1:0]存放低2位ADC结果

1：ADC_RES[1:0]存放高2位ADC结果，ADC_RESL[7:0]存放低8位ADC结果

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

STC MCU Limited

2.1.3 可编程时钟输出(也可作分频器使用)

STC15系列单片机最多有六路可编程时钟输出(如STC15W4K60S4系列),这六路可编程时钟输出分别是: MCLKO/P5.4或MCLKO_2/XTAL2/P1.6, T0CLKO/P3.5, T1CLKO/P3.4, T2CLKO/P3.0, T3CLKO/P0.4, T4CLKO/P0.6。对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz；对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz。

STC15全系列的可编程时钟输出的类型如下表所示。

可编程时钟输出 / 单片机型号	主时钟输出(MCLKO/P5.4)	定时器/计数器0时钟输出(T0CLKO/P3.5)	定时器/计数器1时钟输出(T1CLKO/P3.4)	定时器/计数器2时钟输出(T2CLKO/P3.0)	定时器/计数器3时钟输出(T3CLKO/P.0.4)	定时器/计数器4时钟输出(T4CLKO/P0.6)
STC15F101W系列	该系列主时钟输出在MCLKO/P3.4	√		√		
STC15F408AD系列	√	√		√		
STC15W201S系列	√	√		√		
STC15W408AS系列	√ (该系列主时钟输出还可在MCLKO_2/XTAL2/P1.6)	√		√		
STC15W408S系列	√ (该系列主时钟输出还可在MCLKO_2/P1.6)	√	√	√		
STC15W1K16S系列	√ (该系列主时钟输出还可在MCLKO_2/XTAL2/P1.6)	√	√	√		
STC15F2K60S2系列	√	√	√	√		
STC15W4K60S4系列	√ (该系列主时钟输出还可在MCLKO_2/XTAL2/P1.6)	√	√	√	√	√

上表中√表示对应的系列有相应的可编程时钟输出。

特别注意：对于STC15W1K16S系列和STC15W408S单片机，若要使用T0CLKO时钟输出功能，必须将P3.5口设置为强推挽输出模式。

STC MCU Limited

2.1.3.1 与可编程时钟输出有关的特殊功能寄存器

符号	描述	地址	位地址及其符号 MSB LSB	复位值
AUXR	辅助寄存器	8EH	T0x12 | T1x12 | UART_M0x6 | T2R | T2_C/$\overline{T}$ | T2x12 | EXTRAM | S1ST2	0000 0001B
INT_CLKO AUXR2	External Interrupt enable and Clock output register	8FH	- | EX4 | EX3 | EX2 | - | T2CLKO | T1CLKO | T0CLKO	x000 x000B
CLK_DIV (PCON2)	时钟分配器	97H	MCKO_S1 | MCKO_S1 | ADRJ | Tx_Rx | MCLKO_2 | CLKS2 | CLKS1 | CLKS0	0000 0000B
T4T3M	T4和T3的控制寄存器	D1H	T4R | T4_C/$\overline{T}$ | T4x12 | T4CLKO | T3R | T3_C/$\overline{T}$ | T3x12 | T3CLKO	0000 0000B

特殊功能寄存器INT_CLKO/AUXR/CLK_DIV/T4T3M的C语言声明：

```
sfr     INT_CLKO    =  0x8F;     //新增加的特殊功能寄存器INT_CLKO的地址声明
sfr     AUXR        =  0x8E;     //特殊功能寄存器AUXR的地址声明
sfr     CLK_DIV     =  0x97;     //特殊功能寄存器CLK_DIV的地址声明
sfr     T4T3M       =  0xD1;     //新增加的特殊功能寄存器T4T3M的地址声明
```

特殊功能寄存器INT_CLKO/AUXR/CLK_DIV/T4T3M的汇编语言声明：

```
INT_CLKO    EQU  8FH    ;新增加的特殊功能寄存器INT_CLKO的地址声明
AUXR        EQU  8EH    ;特殊功能寄存器AUXR的地址声明
CLK_DIV     EQU  97H    ;特殊功能寄存器CLK_DIV的地址声明
T4T3M       EQU  D1H    ;新增加的特殊功能寄存器T4T3M的地址声明
```

1. CLK_DIV (PCON2) : 时钟分频寄存器(不可位寻址)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， 输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， 输出时钟频率 = MCLK / 4

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK是指主时钟频率。

STC15W4K60S4系列单片机在MCLKO/P5.4口或MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC Mcu Limited

STC15W系列单片机通过CLK_DIV.3/MCLKO_2位来选择是在MCLKO/P5.4口对外输出主时钟，还是在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位

0：在MCLKO/P5.4口对外输出主时钟；

1：在MCLKO_2/XTAL2/P1.6口对外输出主时钟；

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

ADRJ：ADC转换结果调整

0：ADC_RES[7:0]存放高8位ADC结果，ADC_RESL[1:0]存放低2位ADC结果

1：ADC_RES[1:0]存放高2位ADC结果，ADC_RESL[7:0]存放低8位ADC结果

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 （系统时钟是指对主时钟进行分频后供给CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D转换的实际工作时钟）
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

STC MCU Limited

2. INT_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

B0 – T0CLKO ： 是否允许将P3.5/T1脚配置为定时器0(T0)的时钟输出T0CLKO

1，将P3.5/T1管脚配置为定时器0的时钟输出T0CLKO，输出时钟频率=T0溢出率/2

若定时器/计数器T0工作在定时器模式0(16位自动重装载模式)时，

如果C/$\overline{T}$=0，定时器/计数器T0是对内部系统时钟计数，则：

T0工作在1T模式(AUXR.7/T0x12=1)时的输出频率 = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH0, RL_TL0])/2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

若定时器/计数器T0工作在定时器模式2(8位自动重装模式)，

如果C/$\overline{T}$=0, 定时器/计数器T0是对内部系统时钟计数，则：

T0工作在1T模式(AUXR.7/T0x12=1)时的输出频率 = (SYSclk) / (256-TH0) / 2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出频率 = (SYSclk) / 12 / (256-TH0) / 2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (256-TH0) / 2

0，不允许P3.5/T1管脚被配置为定时器0的时钟输出

B1 – T1CLKO：是否允许将P3.4/T0脚配置为定时器1(T1)的时钟输出T1CLKO

1，将P3.4/T0管脚配置为定时器1的时钟输出T1CLKO，输出时钟频率= T1溢出率/2

若定时器/计数器T1工作在定时器模式0(16位自动重装载模式)，

如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH1, RL_TL1])/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH1, RL_TL1])/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

若定时器/计数器T1工作在模式2(8位自动重装模式)，

如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (256-TH1)/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk)/12/(256-TH1)/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1) / 2

0，不允许P3.4/T0管脚被配置为定时器1的时钟输出

B2 - T2CLKO：是否允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

1：允许将P3.0脚配置为定时器2的时钟输出T2CLKO，输出时钟频率=T2溢出率/2

如果T2_C/$\overline{T}$=0，定时器/计数器T2是对内部系统时钟计数，则：

T2工作在1T模式(AUXR.2/T2x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH2, RL_TL2])/2

T2工作在12T模式(AUXR.2/T2x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH2, RL_TL2])/2

如果T2_C/$\overline{T}$=1，定时器/计数器T2是对外部脉冲输入(P3.1/T2)计数，则：

输出时钟频率 = (T2_Pin_CLK) / (65536-[RL_TH2, RL_TL2])/2

0： 不允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

B4 - EX2：允许外部中断2(INT2)

B5 - EX3：允许外部中断3(INT3)

B6 - EX4：允许外部中断4(INT4)

3、辅助特殊功能寄存器：AUXR(地址：0x8E)

AUXR : Auxiliary register(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

B7 - T0x12：定时器0速度控制位。

0：定时器0速度是传统8051单片机定时器的速度，即12分频；

1：定时器0速度是传统8051单片机定时器速度的12倍，即不分频。

B6 - T1x12：定时器1速度控制位。

0：定时器1速度是传统8051单片机定时器的速度，即12分频；

1：定时器1速度是传统8051单片机定时器速度的12倍，即不分频。

如果串口1用T1作为波特率发生器，则由T1x12位决定串口1是12T还是1T。

B5 - UART_M0x6：串口1模式0的通信速度设置位。

0：串口1模式0的速度是传统8051单片机串口的速度，即12分频；

1：串口1模式0的速度是传统8051单片机串口速度的6倍，即2分频。

B4 - T2R：定时器2运行控制位。

0：不允许定时器2运行；

1：允许定时器2运行。

B3 - T2_C/T：控制定时器2用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T2/P3.1的外部脉冲进行计数)

B2 - T2x12：定时器2速度控制位

0，定时器2是传统8051单片机的速度，12分频；

1，定时器2的速度是传统8051单片机速度的12倍，不分频

如果串口1或串口2用T2作为波特率发生器，则由T2x12决定串口1或串口2是12T还是1T.

B1 - EXTRAM：内部/外部RAM存取控制位。

0：允许使用逻辑上在片外、物理上在片内的扩展RAM；

1：禁止使用逻辑上在片外、物理上在片内的扩展RAM。

B0 - S1ST2：串口1(UART1)选择定时器2作波特率发生器的控制位。

0：选择定时器1作为串口1(UART1)的波特率发生器；

1：选择定时器2作为串口1(UART1)的波特率发生器，此时定时器1得到释放，可以作为独立定时器使用。

4、定时器T4和T3的控制寄存器：T4T3M(地址：0xD1)

T4T3M(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO

B7 - T4R：定时器4运行控制位。

0：不允许定时器4运行；

1：允许定时器4运行。

B6 - T4_C/$\overline{T}$：控制定时器4用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T4/P0.7的外部脉冲进行计数)

B5 - T4x12：定时器4速度控制位。

0：定时器4速度是8051单片机定时器的速度，即12分频；

1：定时器4速度是8051单片机定时器速度的12倍，即不分频。

B4 - T4CLKO：是否允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

1：允许将P0.6脚配置为定时器4的时钟输出T4CLKO，输出时钟频率=T4溢出率/2

如果T4_C/$\overline{T}$=0，定时器/计数器T4是对内部系统时钟计数，则：

T4工作在1T模式(T4T3M.5/T4x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH4, RL_TL4])/2

T4工作在12T模式(T4T3M.5/T4x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH4, RL_TL4])/2

如果T4_C/$\overline{T}$=1，定时器/计数器T4是对外部脉冲输入(P0.7/T4)计数，则：

输出时钟频率 = (T4_Pin_CLK) / (65536-[RL_TH4, RL_TL4])/2

0：不允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

B3 - T3R：定时器3运行控制位。

0：不允许定时器3运行；

1：允许定时器3运行。

B2 - T3_C/$\overline{T}$：控制定时器3用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T3/P0.5的外部脉冲进行计数)

B1 - T3x12：定时器3速度控制位。

0：定时器3速度是8051单片机定时器的速度，即12分频；

1：定时器3速度是8051单片机定时器速度的12倍，即不分频。

B0 - T3CLKO：是否允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

1：允许将P0.4脚配置为定时器3的时钟输出T3CLKO，输出时钟频率=T3溢出率/2

如果T3_C/$\overline{T}$=0，定时器/计数器T3是对内部系统时钟计数，则：

T3工作在1T模式(T4T3M.1/T3x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH3, RL_TL3])/2

T3工作在12T模式(T4T3M.1/T3x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH3, RL_TL3])/2

如果T3_C/$\overline{T}$=1，定时器/计数器T3是对外部脉冲输入(P0.5/T3)计数，则：

输出时钟频率 = (T3_Pin_CLK) / (65536-[RL_TH3, RL_TL3])/2

0：不允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

STC MCU Limited

2.1.3.2 主时钟输出及测试程序(C和汇编)

主时钟可以是内部高精度R/C时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz，如果频率过高，需进行分频输出；而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz，如果频率过高，需进行分频输出。如果频率过高，需进行分频输出。

CLK_DIV (PCON2)：时钟分频寄存器(不可位寻址)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

如何利用MCLKO/P5.4或MCLKO_2/XTAL2/P1.6管脚输出时钟

MCLKO/P5.4或MCLKO_2/XTAL2/P1.6的时钟输出控制由CLK_DIV寄存器的MCKO_S1和MCKO_S0位控制。通过设置MCKO_S1(CLK_DIV.7)和MCKO_S0(CLK_DIV.6)可将MCLKO/P5.4管脚配置为主时钟输出同时还可以设置该主时钟的输出频率。

特殊功能寄存器：CLK_DIV（地址：97H）

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， **输出时钟频率** = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， **输出时钟频率** = MCLK / 4

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK是指主时钟频率。

STC15W4K60S4系列单片机在MCLKO/P5.4口或MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15W系列单片机通过CLK_DIV.3/MCLKO_2位来选择是在MCLKO/P5.4口对外输出主时钟，还是在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位

0：在MCLKO/P5.4口对外输出主时钟；

1：在MCLKO_2/XTAL2/P1.6口对外输出主时钟；

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz，如果频率过高，需进行分频输出。

而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz，如果频率过高，需进行分频输出。

STC MCU Limited

下面是主时钟输出的示例程序：

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ----------------------------------------------------------------------*/
/* --- 演示STC15F2K60S2 系列单片机的主时钟输出 ----------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ---------------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ------------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;

#define   FOSC    18432000L
//-----------------------------------------
sfr       CLK_DIV       =       0x97;                 //时钟分频寄存器

//-----------------------------------------

void main()
{
          CLK_DIV       =       0x40;                 //0100,0000 P5.4输出频率为SYSclk
//        CLK_DIV       =       0x80;                 //1000,0000 P5.4输出频率为SYSclk/2
//        CLK_DIV       =       0xC0;                 //1100,0000 P5.4输出频率为SYSclk/4

          while (1);                                  //程序终止
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ----------------------------------------------------------------------*/
/* --- 演示STC15F2K60S2 系列单片机的主时钟输出 ----------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ---------------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

CLK_DIV         DATA    097H                    //IRC时钟输出控制寄存器

;-----------------------------------------
;interrupt vector table

        ORG     0000H
        LJMP    MAIN                            //复位入口
;-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,             #3FH            //initial SP
        MOV     CLK_DIV,        #40H            //0100,0000 P5.4输出频率为SYSclk
//      MOV     CLK_DIV,        #80H            //1000,0000 P5.4输出频率为SYSclk/2
//      MOV     CLK_DIV,        #C0H            //1100,0000 P5.4输出频率为SYSclk/4
        SJMP    $

//-----------------------------------------
        END
```

STC MCU Limited

2.1.3.3 定时器0对系统时钟或外部引脚T0的时钟输入进行可编程分频输出及测试程序

如何利用T0CLKO/P3.5管脚输出时钟

T0CLKO/P3.5管脚是否输出时钟由INT_CLKO (AUXR2)寄存器的T0CLKO位控制

AUXR2.0 - T0CLKO :　1, 允许时钟输出

0, 禁止时钟输出

T0CLKO的输出时钟频率由定时器0控制, 相应的定时器0需要工作在定时器的模式0(16位自动重装载模式)或模式2(8位自动重装载模式), 不要允许相应的定时器中断, 免得CPU反复进中断, 当然在特殊情况下也可允许相应的定时器中断。

新增加的特殊功能寄存器: INT_CLKO (AUXR2)(地址：0x8F)

当T0CLKO/INT_CLKO.0=1时，P3.5/T1管脚配置为定时器0的时钟输出T0CLKO。

输出时钟频率 = T0 溢出率/2

若定时器/计数器T0工作在定时器模式0(16位自动重装载模式)时，(如下图所示)

如果C/$\overline{T}$=0，定时器/计数器T0对内部系统时钟计数，则：

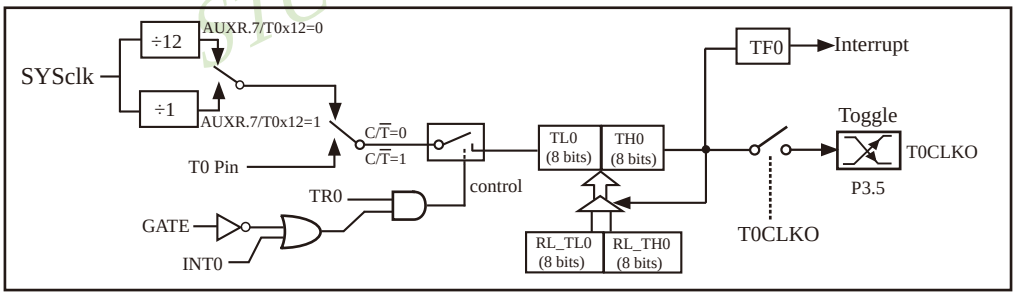
T0工作在1T模式(AUXR.7/T0x12=1)时的输出时钟频率 = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH0, RL_TL0])/2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

RL_TH0为TH0的重装载寄存器，RL_TL0为TL0的重装载寄存器。



定时器/计数器0的模式 0: 16位自动重装

STC创新设计，请不要再抄袭，再抄袭就很无耻了

当T0CLKO/INT_CLKO.0=1且定时器/计数器T0工作在定时器模式2(8位自动重装模式)时，(如下图所示)

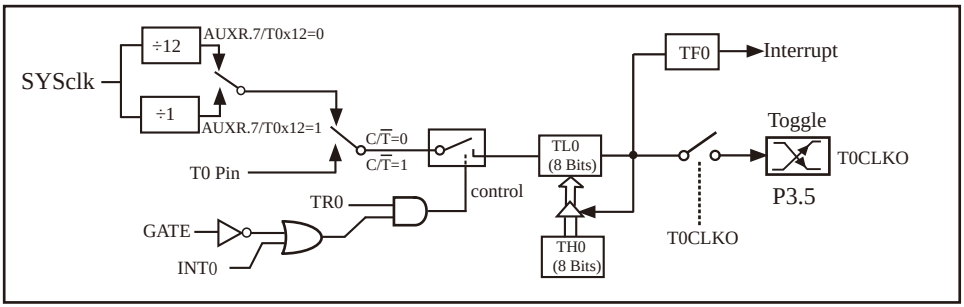
如果C/$\overline{T}$=0，定时器/计数器T0对内部系统时钟计数，则：

T0工作在1T模式(AUXR.7/T0x12=1)时的输出时钟频率=(SYSclk) / (256-TH0) / 2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出时钟频率=(SYSclk) / 12 / (256-TH0) / 2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (256-TH0) / 2



定时器/计数器0的模式 2: 8位自动重装

特别注意：对于STC15W1K16S系列和STC15W408S单片机，若要使用T0CLKO时钟输出功能，必须将P3.5口设置为强推挽输出模式。

下面是定时器0对内部系统时钟或外部引脚T0/P3.4的时钟输入进行可编程时钟分频输出的程序举例(C和汇编)：

1. C程序：

```
/*-----------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------------*/
/* --- 演示STC15F2K60S2 系列单片机定时器0的可编程时钟分频输出-----*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*-----------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define FOSC 18432000L
//-----------------------------------------------
sfr     AUXR        =       0x8e;                   //辅助特殊功能寄存器
sfr     INT_CLKO    =       0x8f;                   //唤醒和时钟输出功能寄存器

sbit    T0CLKO      =       P3^5;                   //定时器0的时钟输出脚

#define F38_4KHz    (65536-FOSC/2/38400)            //1T模式
//#define F38_4KHz  (65536-FOSC/2/12/38400)         //12T模式
```

STC MCU Limited

```
//-----------------------------------------------
void main()
{
        AUXR    |=      0x80;                   //定时器0为1T模式
//      AUXR    &=      ~0x80;                  //定时器0为12T模式

        TMOD    =       0x00;                   //设置定时器为模式0(16位自动重装载)

        TMOD    &=      ~0x04;                  //C/T0=0, 对内部时钟进行时钟输出
//      TMOD    |=      0x04;                   //C/T0=1, 对T0引脚的外部时钟进行时钟输出

        TL0     =       F38_4KHz;               //初始化计时值
        TH0     =       F38_4KHz >> 8;
        TR0     =       1;
        INT_CLKO        =       0x01;           //使能定时器0的时钟输出功能

        while (1);                              //程序终止
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------*/
/* --- 演示STC15F2K60S2 系列单片机定时器0的可编程时钟分频输出-----*/
/* --- 研发顾问QQ：800003751-------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------*/
/* --- Tel: 86-755-82948412 --------------------------------------------------*/
/* --- Web: www.STCMCU.com ---------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*-------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR            DATA    08EH            //辅助特殊功能寄存器
INT_CLKO        DATA    08FH            //唤醒和时钟输出功能寄存器

T0CLKO          BIT     P3.5            //定时器0的时钟输出脚

F38_4KHz        EQU     0FF10H          //38.4KHz(1T模式下, 65536-18432000/2/38400)
//F38_4KHz      EQU     0FFECH          //38.4KHz(12T模式下, (65536-18432000/2/12/38400)
//-----------------------------------------------
```

STC MCU Limited

```
        ORG     0000H
        LJMP    MAIN                            //复位入口

//---------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #80H                    //定时器0为1T模式
//      ANL     AUXR,   #7FH                    //定时器0为12T模式

        MOV     TMOD,   #00H                    //设置定时器为模式0(16位自动重装载)

        ANL     TMOD,   #0FBH                   //C/T0=0, 对内部时钟进行时钟输出
//      ORL     TMOD,   #04H                    //C/T0=1, 对T0引脚的外部时钟进行时钟输出

        MOV     TL0,    #LOW F38_4KHz           //初始化计时值
        MOV     TH0,    #HIGH F38_4KHz
        SETB    TR0
        MOV     INT_CLKO,       #01H            //使能定时器0的时钟输出功能

        SJMP    $                               //程序终止

;---------------------------------------

        END
```

STC MCU Limited

2.1.3.4 定时器1对系统时钟或外部引脚T1的时钟输入进行可编程分频输出及测试程序

如何利用T1CLKO/P3.4管脚输出时钟

T1CLKO/P3.4管脚是否输出时钟由INT_CLKO (AUXR2)寄存器的T1CLKO位控制

AUXR2.1 - T1CLKO :　1, 允许时钟输出

0, 禁止时钟输出

T1CLKO的输出时钟频率由定时器1控制, 相应的定时器1需要工作在定时器的模式0(16位自动重装载模式)或模式2(8位自动重装载模式), 不要允许相应的定时器中断, 免得CPU反复进中断, 当然在特殊情况下也可允许相应的定时器中断。

新增加的特殊功能寄存器: INT_CLKO (AUXR2)(地址：0x8F)

当T1CLKO/INT_CLKO.1=1时，P3.4/T0管脚配置为定时器1的时钟输出T1CLKO。

输出时钟频率 = T1 溢出率/2

若定时器/计数器T1工作在定时器模式0(16位自动重装载模式)时，(如下图所示)

如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

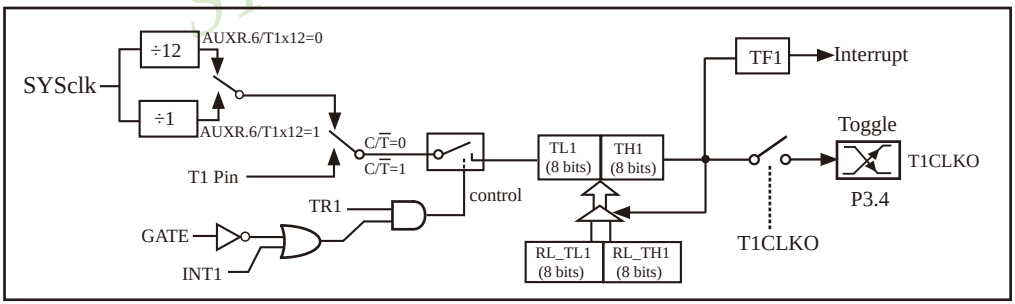
T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH1, RL_TL1])/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH1, RL_TL1])/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

RL_TH1为TH1的重装载寄存器，RL_TL1为TL1的重装载寄存器。



定时器/计数器1的模式 0: 16位自动重装

STC创新设计，请不要再抄袭，再抄袭就很无耻了

当T1CLKO/INT_CLKO.1=1且定时器/计数器T1工作在定时器模式2(8位自动重装模式)时，(如下图所示)

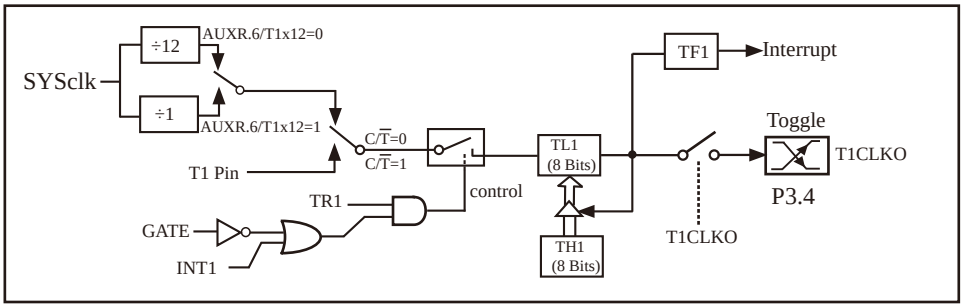
如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (256-TH1)/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk)/12/(256-TH1)/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1) / 2



定时器/计数器1的模式 2: 8位自动重装

下面是定时器1对内部系统时钟或外部引脚T1/P3.5的时钟输入进行可编程时钟分频输出的程序举例(C和汇编):

1. C程序:

```
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机定时器1的可编程时钟分频输出----------------------------------*/
/* --- 研发顾问QQ：800003751--------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 --------------------------------------------------------------*/
/* --- Web: www.STCMCU.com ---------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 --------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;

#define   FOSC    18432000L

//-----------------------------------------------
sfr   AUXR        =       0x8e;                   //辅助特殊功能寄存器
sfr   INT_CLKO    =       0x8f;                   //唤醒和时钟输出功能寄存器

sbit  T1CLKO      =       P3^4;                   //定时器1的时钟输出脚

#define   F38_4KHz        (65536-FOSC/2/38400)    //1T模式
//#define F38_4KHz        (65536-FOSC/2/12/38400) //12T模式
```

```
//-------------------------------------------
void main()
{
        AUXR    |=      0x40;                   //定时器1为1T模式
//      AUXR    &=      ~0x40;                  //定时器1为12T模式

        TMOD    =       0x00;                   //设置定时器为模式0(16位自动重装载)

        TMOD    &=      ~0x40;                  //C/T1=0, 对内部时钟进行时钟输出
//      TMOD    |=      0x40;                   //C/T1=1, 对T1引脚的外部时钟进行时钟输出

        TL1     =       F38_4KHz;               //初始化计时值
        TH1     =       F38_4KHz >> 8;
        TR1     =       1;
        INT_CLKO        =       0x02;           //使能定时器1的时钟输出功能

        while (1);                              //程序终止
}
```

STC MCU Limited

2. 汇编程序：

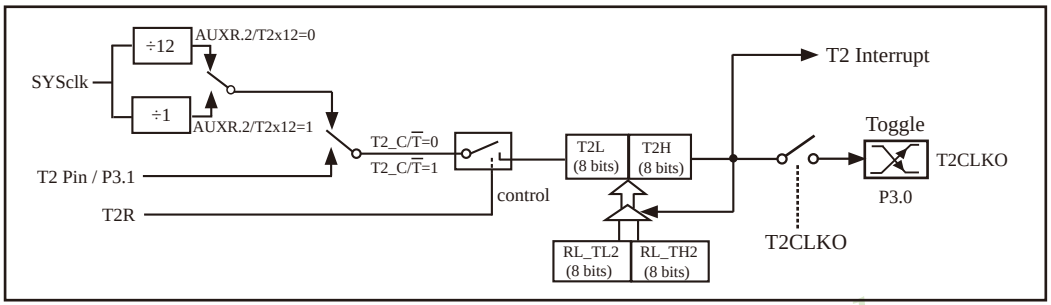
```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ----------------------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机定时器1的可编程时钟分频输出---------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ---------------------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR            DATA    08EH            //辅助特殊功能寄存器
INT_CLKO        DATA    08FH            //唤醒和时钟输出功能寄存器

T1CLKO          BIT     P3.4            //定时器1的时钟输出脚

F38_4KHz        EQU     0FF10H          //38.4KHz(1T模式下, 65536-18432000/2/38400)
//F38_4KHz      EQU     0FFECH          //38.4KHz(12T模式下, (65536-18432000/2/12/38400)
```

```
        ORG     0000H
        LJMP    MAIN                        //复位入口

//-----------------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,  #40H                 //定时器1为1T模式
//      ANL     AUXR,  #0BFH                //定时器1为12T模式

        MOV     TMOD,  #00H                 //设置定时器为模式0(16位自动重装载)

        ANL     TMOD,  #0BFH                //C/T1=0, 对内部时钟进行时钟输出
//      ORL     TMOD,  #40H                 //C/T1=1, 对T1引脚的外部时钟进行时钟输出

        MOV     TL1,    #LOW F38_4KHz       //初始化计时值
        MOV     TH1,    #HIGH F38_4KHz
        SETB    TR1
        MOV     INT_CLKO,       #02H        //使能定时器1的时钟输出功能

        SJMP    $                           //程序终止

;-----------------------------------------------

        END
```

STC MCU Limited

2.1.3.5 定时器2对系统时钟或外部引脚T2的时钟输入进行可编程分频输出及测试程序

T2可以当定时器用，也可以当串口的波特率发生器和可编程时钟输出。
定时器2的原理框图如下：



定时器/计数器2的工作模式: 16位自动重装载

STC创新设计，请不要再抄袭，再抄袭就很无耻了

如何利用T2CLKO/P3.0管脚输出时钟

AUXR2.2 - T2CLKO：是否允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

1： 允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO，

0： 不允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

当T2CLKO/INT_CLKO.2=1时，P3.0管脚配置为定时器2的时钟输出T2CLKO。

输出时钟频率 = T2 溢出率/2

如果T2_C/$\overline{T}$=0，定时器/计数器T2对内部系统时钟计数，则：

T2工作在1T模式(AUXR.2/T2x12=1)时的输出时钟频率 = (SYSclk)/(65536-[RL_TH2, RL_TL2])/2

T2工作在12T模式(AUXR.2/T2x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH2, RL_TL2])/2

如果T2_C/$\overline{T}$=1，定时器/计数器T2是对外部脉冲输入(P3.1/T2)计数，则：

输出时钟频率 = (T2_Pin_CLK) / (65536-[RL_TH2, RL_TL2])/2

RL_TH2为T2H的重装载寄存器，RL_TL2为T2L的重装载寄存器。

用户在程序中如何具体设置T2CLKO/P3.0管脚输出时钟

1. 对定时器2寄存器T2H/T2L送16位重装载值, [T2H,T2L] = #reload_data
2. 对AUXR寄存器中的T2R位置1, 让定时器2运行
3. 对AUXR2/INT_CLKO寄存器中的T2CLKO位置1, 让定时器2的溢出在P3.0口输出时钟。

注意：当定时器/计数器2用作可编程时钟输出时，不要允许相应的定时器中断, 免得CPU反复进中断，在特殊情况下也可允许定时器/计数器2中断。

下面是定时器2对内部系统时钟或外部引脚T2/P3.1的时钟输入进行可编程时钟分频输出的程序举例(C和汇编):

1. C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/* --- STC15F2K60S2  系列  定时器2的可编程时钟分频输出举例-----------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef unsigned char       BYTE;
typedef unsigned int        WORD;

#define FOSC 18432000L

//-----------------------------------------

sfr     AUXR            = 0x8e;             //辅助特殊功能寄存器
sfr     INT_CLKO        = 0x8f;             //唤醒和时钟输出功能寄存器
sfr     T2H             = 0xD6;             //定时器2高8位
sfr     T2L             = 0xD7;             //定时器2低8位

sbit    T2CLKO          = P3^0;             //定时器2的时钟输出脚

#define F38_4KHz        (65536-FOSC/2/38400)        //1T模式
//#define F38_4KHz      (65536-FOSC/2/12/38400)     //12T模式

//-----------------------------------------

void main()
{
        AUXR    |=      0x04;               //定时器2为1T模式
//      AUXR    &=      ~0x04;              //定时器2为12T模式
```

STC MCU Limited

```
    AUXR    &=      ~0x08;          //T2_C/T=0, 对内部时钟进行时钟输出
//  AUXR    |=      0x08;           //T2_C/T=1, 对T2(P3.1)引脚的外部时钟进行时钟输出

    T2L     =       F38_4KHz;                   //初始化计时值
    T2H     =       F38_4KHz >> 8;

    AUXR    |=      0x10;                       //定时器2开始计时
    INT_CLKO        =       0x04;               //使能定时器2的时钟输出功能

    while (1);                                  //程序终止
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2  系列  定时器2可编程时钟分频输出举例-------------------------*/
/* --- 研发顾问QQ：  800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码, 请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码, 请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR            DATA    08EH            //辅助特殊功能寄存器
INT_CLKO        DATA    08FH            //唤醒和时钟输出功能寄存器
T2H             DATA    0D6H            //定时器2高8位
T2L             DATA    0D7H            //定时器2低8位

T2CLKO          BIT     P3.0            //定时器2的时钟输出脚

F38_4KHz        EQU     0FF10H          //38.4KHz(1T模式下，  65536-18432000/2/38400)
//F38_4KHz      EQU     0FFECH          //38.4KHz(12T模式下, (65536-18432000/2/12/38400)

//-----------------------------------------------
```

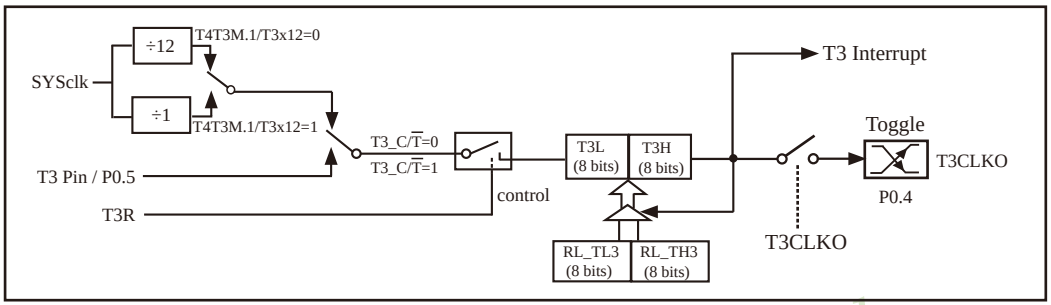
STC MCU Limited

```
        ORG     0000H
        LJMP    MAIN                                    //复位入口

//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #04H                            //定时器2为1T模式
//      ANL     AUXR,   #0FBH                           //定时器2为12T模式

        ANL     AUXR,   #0F7H           //T2_C/T=0, 对内部时钟进行时钟输出
//      ORL     AUXR,   #08H            //T2_C/T=1, 对T2(P3.1)引脚的外部时钟进行时钟输出

        MOV     T2L,    #LOW F38_4KHz                   //初始化计时值
        MOV     T2H,    #HIGH F38_4KHz
        ORL     AUXR,   #10H                            //定时器2开始计时
        MOV     INT_CLKO,   #04H                        //使能定时器2的时钟输出功能

        SJMP    $                                       //程序终止

;-----------------------------------------

        END
```

STC MCU Limited

2.1.3.6 定时器3对系统时钟或外部引脚T3的时钟输入进行可编程分频输出及测试程序

T3可以当定时器用，也可以当串口3的波特率发生器和可编程时钟输出。
定时器3的原理框图如下：



定时器/计数器3的工作模式: 16位自动重装载

STC创新设计，请不要再抄袭，再抄袭就很无耻了

如何利用T3CLKO/P0.4管脚输出时钟

T4T3M.0 - T3CLKO：是否允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

1： 允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO，

0： 不允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

当T3CLKO/T4T3M.0=1时，P0.4管脚配置为定时器3的时钟输出T3CLKO。

输出时钟频率 = T3 溢出率/2

如果T3_C/$\overline{T}$=0，定时器/计数器T3对内部系统时钟计数，则：

T3工作在1T模式(T4T3M.1/T3x12=1)时的输出时钟频率 = (SYSclk)/(65536-[RL_TH3, RL_TL3])/2

T3工作在12T模式(T4T3M.1/T3x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH3, RL_TL3])/2

如果T3_C/$\overline{T}$=1，定时器/计数器T3是对外部脉冲输入(P0.5/T3)计数，则：

输出时钟频率 = (T3_Pin_CLK) / (65536-[RL_TH3, RL_TL3])/2

RL_TH3为T3H的重装载寄存器，RL_TL3为T3L的重装载寄存器。

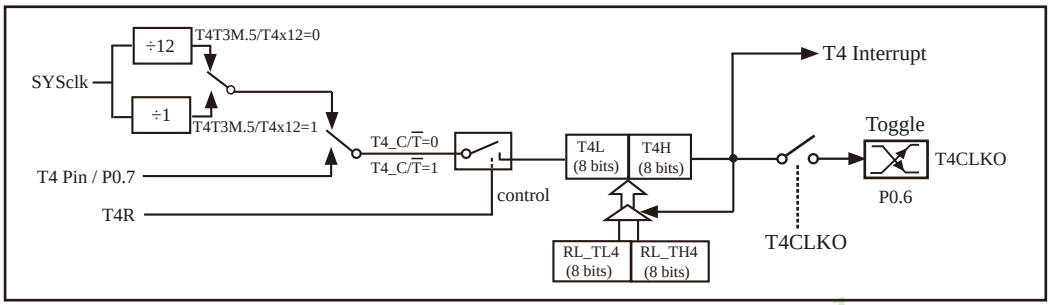
用户在程序中如何具体设置T3CLKO/P0.4管脚输出时钟

1. 对定时器3寄存器T3H/T3L送16位重装载值, [T3H,T3L] = #reload_data
2. 对T4T3M寄存器中的T3R位置1, 让定时器3运行
3. 对T4T3M寄存器中的T3CLKO位置1, 让定时器3的溢出在P0.4口输出时钟。

注意：当定时器/计数器3用作可编程时钟输出时，不要允许相应的定时器中断, 免得CPU反复进中断，在特殊情况下也可允许定时器/计数器3中断。

2.1.3.7 定时器4对系统时钟或外部引脚T4的时钟输入进行可编程分频输出及测试程序

T4可以当定时器用，也可以当串口4的波特率发生器和可编程时钟输出。
定时器4的原理框图如下：



定时器/计数器4的工作模式: 16位自动重装载

STC创新设计，请不要再抄袭，再抄袭就很无耻了

如何利用T4CLKO/P0.6管脚输出时钟

T4T3M.4 - T4CLKO：是否允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

1： 允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO，

0： 不允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

当T4CLKO/T4T3M.4=1时，P0.6管脚配置为定时器4的时钟输出T4CLKO。

输出时钟频率 = T4 溢出率/2

如果T4_C/$\overline{T}$=0，定时器/计数器T4对内部系统时钟计数，则：

T4工作在1T模式(T4T3M.5/T4x12=1)时的输出时钟频率 = (SYSclk)/(65536-[RL_TH4, RL_TL4])/2

T4工作在12T模式(T4T3M.5/T4x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH4, RL_TL4])/2

如果T4_C/$\overline{T}$=1，定时器/计数器T4是对外部脉冲输入(P0.7/T4)计数，则：

输出时钟频率 = (T4_Pin_CLK) / (65536-[RL_TH4, RL_TL4])/2

RL_TH4为T4H的重装载寄存器，RL_TL4为T4L的重装载寄存器。

用户在程序中如何具体设置T4CLKO/P0.6管脚输出时钟

1. 对定时器4寄存器T4H/T4L送16位重装载值, [T4H,T4L] = #reload_data
2. 对T4T3M寄存器中的T4R位置1, 让定时器4运行
3. 对T4T3M寄存器中的T4CLKO位置1, 让定时器4的溢出在P0.6口输出时钟。

注意：当定时器/计数器4用作可编程时钟输出时，不要允许相应的定时器中断, 免得CPU反复进中断，在特殊情况下也可允许定时器/计数器4中断。

2.2 复位

STC15系列单片机有7种复位方式：外部RST引脚复位，软件复位，掉电复位/上电复位(并可选择增加额外的复位延时180mS，也叫MAX810专用复位电路，其实就是在上电复位后增加一个180mS复位延时)，内部低压检测复位，MAX810专用复位电路复位，看门狗复位以及程序地址非法复位。

2.2.1 外部RST引脚复位

STC15F101W系列单片机的复位管脚在RST/P3.4口，其他STC15系列单片机的复位管脚均在RST/P5.4口。下面以P5.4/RST为例介绍外部RST引脚的复位。

外部RST引脚复位就是从外部向RST引脚施加一定宽度的复位脉冲，从而实现单片机的复位。P5.4/RST管脚出厂时被配置为I/O口，要将其配置为复位管脚，可在ISP烧录程序时设置。如果P5.4/RST管脚已在ISP烧录程序时被设置为复位脚，那P5.4/RST就是芯片复位的输入脚。将RST复位管脚拉高并维持至少24个时钟加20us后,单片机会进入复位状态，将RST复位管脚拉回低电平后，单片机结束复位状态并将特殊功能寄存器IAP_CONTR中的SWBS/IAP_CONTR.6位置1，同时从系统ISP监控程序区启动。外部RST引脚复位是热启动复位中的硬复位。

2.2.2 软件复位及其测试程序(C和汇编)

用户应用程序在运行过程当中，有时会有特殊需求，需要实现单片机系统软复位（热启动复位中的软复位之一），传统的8051单片机由于硬件上未支持此功能，用户必须用软件模拟实现，实现起来较麻烦。现STC新推出的增强型8051根据客户要求增加了IAP_CONTR特殊功能寄存器,实现了此功能。用户只需简单的控制IAP_CONTR特殊功能寄存器的其中两位 SWBS/SWRST 就可以实现系统复位了。

IAP_CONTR: ISP/IAP 控制寄存器

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	name	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0

IAPEN: ISP/IAP功能允许位。0：禁止IAP读/写/擦除Data Flash/EEPROM
1：允许IAP读/写/擦除Data Flash/EEPROM

SWBS：软件选择复位后从用户应用程序区启动(送0)，还是从系统ISP监控程序区启动(送1)。要与SWRST直接配合才可以实现

SWRST：0：不操作； 1：软件控制产生复位，单片机自动复位。

CMD_FAIL：如果IAP地址(由IAP地址寄存器IAP_ADDRH和IAP_ADDRL的值决定)指向了非法地址或无效地址，且送了ISP/IAP命令,并对IAP_TRIG送5Ah/A5h触发失败，则CMD_FAIL为1,需由软件清零。

```
;从用户应用程序区(AP区)软件复位并切换到用户应用程序区(AP区)开始执行程序
MOV IAP_CONTR, #00100000B ;SWBS = 0(选择AP区), SWRST = 1(软复位)
;从系统ISP监控程序区软件复位并切换到用户应用程序区(AP区)开始执行程序
MOV IAP_CONTR, #00100000B ;SWBS = 0(选择AP区), SWRST = 1(软复位)
;从用户应用程序区(AP区)软件复位并切换到系统ISP 监控程序区开始执行程序
MOV IAP_CONTR, #01100000B ;SWBS = 1(选择ISP区), SWRST = 1(软复位)
;从系统ISP 监控程序区软件复位并切换到系统ISP监控程序区开始执行程序
MOV IAP_CONTR, #01100000B ;SWBS = 1(选择ISP区), SWRST = 1(软复位)
```

本复位是整个系统复位，所有的特殊功能寄存器都会复位到初始值，I/O口也会初始化

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 软件复位举例--------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序 */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序 */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------
sfr     IAP_CONTR  =  0xc7;             //IAP控制寄存器

sbit    P10     =     P1^0;
//-----------------------------------------------

void delay()                            //软件延时
{
        int i;

        for (i=0; i<10000; i++)
        {
                _nop_();
                _nop_();
                _nop_();
                _nop_();
        }
}
void main()
{
        P10 = !P10;                     //上电P1.0闪烁一次,便于观察
        delay();
```

STC MCU Limited

```
        P10 = !P10;
        delay();

        IAP_CONTR = 0x20;               //软件复位,系统重新从用户代码区开始运行程序
//      IAP_CONTR = 0x60;               //软件复位,系统重新从ISP代码区开始运行程序

        while (1);
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 软件复位举例--------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

IAP_CONTR   DATA    0C7H                    //IAP控制寄存器
//-----------------------------------------
        ORG     0000H
        LJMP    MAIN                        //复位入口
//-----------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        CPL     P1.0                        //上电P1.0闪烁一次,便于观察
        LCALL   DELAY
        CPL     P1.0
        LCALL   DELAY

        MOV     IAP_CONTR,      #20H        //软件复位,系统重新从用户代码区开始运行程序
//      MOV     IAP_CONTR,      #60H        //软件复位,系统重新从ISP代码区开始运行程序

        JMP     $
;-----------------------------------------
DELAY:
        MOV     R0,     #0                  //软件延时
        MOV     R1,     #0
WAIT:
        DJNZ    R0,     WAIT
        DJNZ    R1,     WAIT
        RET
;-----------------------------------------
        END
```

STC MCU Limited

2.2.3 掉电复位/上电复位

当电源电压VCC低于掉电复位/上电复位检测门槛电压时，所有的逻辑电路都会复位。当内部VCC上升至上电复位检测门槛电压以上后，延迟32768个时钟，掉电复位/上电复位结束。复位状态结束后，单片机将特殊功能寄存器IAP_CONTR中的SWBS/IAP_CONTR.6位置1，同时从系统ISP监控程序区启动。掉电复位/上电复位是冷启动复位之一。

对于5V单片机，它的掉电复位/上电复位检测门槛电压为3.2V；对于3.3V单片机，它的掉电复位/上电复位检测门槛电压为1.8V。

2.2.4 MAX810专用复位电路复位

STC15系列单片机内部集成了MAX810专用复位电路。若MAX810专用复位电路在STC-ISP编程器中被允许，则以后掉电复位/上电复位后将产生约180mS复位延时，复位才被解除。复位解除后单片机将特殊功能寄存器IAP_CONTR中的SWBS/IAP_CONTR.6位置1，同时从系统ISP监控程序区启动。MAX810专用复位电路复位是冷启动复位之一。

2.2.5 内部低压检测复位

除了上电复位检测门槛电压外，STC15单片机还有一组更可靠的内部低压检测门槛电压。当电源电压Vcc低于内部低压检测(LVD)门槛电压时，可产生复位(前提是在STC-ISP编程/烧录用户程序时,允许低压检测复位/禁止低压中断，即将低压检测门槛电压设置为复位门槛电压)。低压检测复位结束后，不影响特殊功能寄存器IAP_CONTR中的SWBS/IAP_CONTR.6位的值，单片机根据复位前SWBS/IAP_CONTR.6的值选择是从用户应用程序区启动，还是从系统ISP监控程序区启动。如果复位前SWBS/IAP_CONTR.6的值为0，则单片机从用户应用程序区启动。反之，如果复位前SWBS/IAP_CONTR.6的值为1，则单片机从系统ISP监控程序区启动。内部低压检测复位是热启动复位中的硬复位之一。

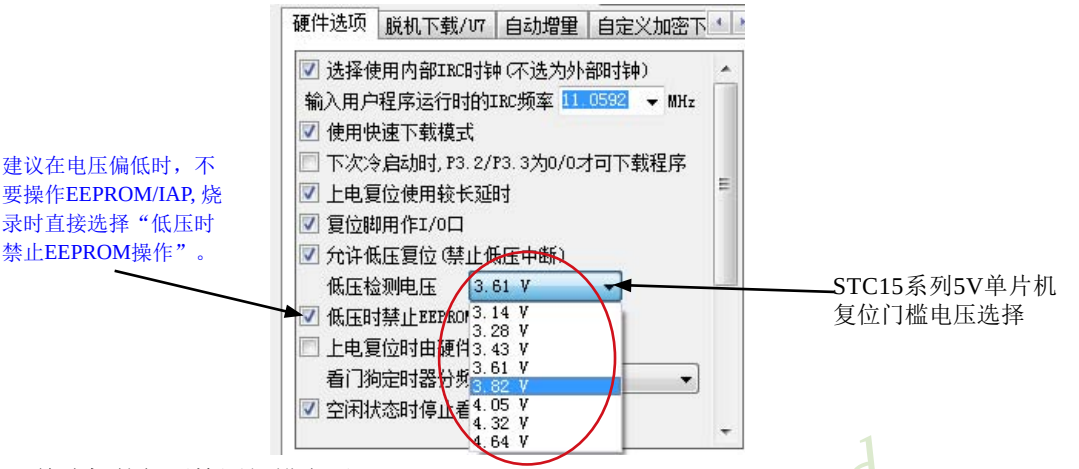
STC15单片机内置了8级可选内部低压检测门槛电压。下表列出了不同温度下STC15系列5V单片机和3.3V单片机所有的低压检测门槛电压。

5V单片机的低压检测门槛电压：

-40 ℃	25 ℃	85 ℃
4.74	4.64	4.60
4.41	4.32	4.27
4.14	4.05	4.00
3.90	3.82	3.77
3.69	3.61	3.56
3.51	3.43	3.38
3.36	3.28	3.23
3.21	3.14	3.09

如果用户所使用的是STC15系列5V单片机，那么用户可以根据单片机的实际工频率在STC-ISP编程器中选择上表中所列出的低压检测门槛电压作为复位门槛电压。如：常温下工作频率是20MHz以上时，可以选择4.32V电压作为复位门槛电压；常温下工作频率是12MHz以下时，可以选择3.82V电压作为复位门槛电压。

STC MCU Limited



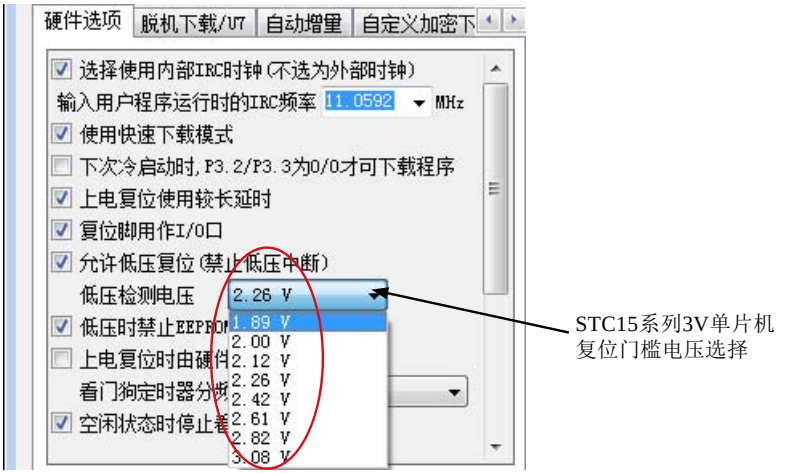
3.3V单片机的低压检测门槛电压：

-40 °C	25 °C	85 °C
3.11	3.08	3.09
2.85	2.82	2.83
2.63	2.61	2.61
2.44	2.42	2.43
2.29	2.26	2.26
2.14	2.12	2.12
2.01	2.00	2.00
1.90	1.89	1.89

STC MCU Limited

如果用户所使用的是STC15系列3.3V单片机，那么用户可以根据单片机的实际工作频率在STC-ISP编程器中选择上表中所列出的低压检测门槛电压作为复位门槛电压。如：常温下工作频率是20MHz以上时，可以选择2.82V电压作为内部低压检测复位门槛电压；常温下工作频率是12MHz以下时，可以选择2.42V电压作为复位门槛电压。

建议在电压偏低时，不要操作EEPROM/IAP, 烧录时直接选择“低压禁止EEPROM操作”。

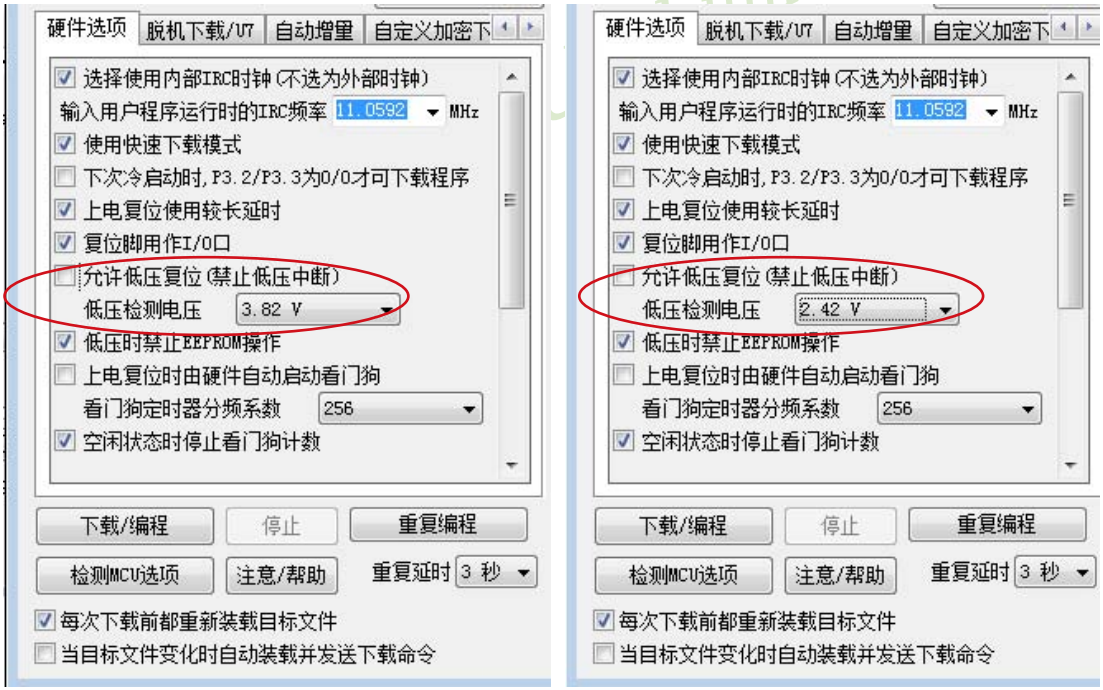


如果在STC-ISP编程/烧录用户应用程序时，不将低压检测设置为低压检测复位，则在用户程序中用户可将低压检测设置为低压检测中断。当电源电压VCC低于内部低压检测(LVD)门槛电压时，低压检测中断请求标志位(LVDF/PCON.5)就会被硬件置位。如果ELVD/IE.6(低压检测中断允许位)被设置为1，低压检测中断请求标志位就能产生一个低压检测中断。

在正常工作和空闲工作状态时，如果内部工作电压Vcc**低于低压检测门槛电压**，低压中断请求标志位(LVDF/PCON.5)自动置1，**与低压检测中断是否被允许无关。即在内部工作电压Vcc**低于低压检测门槛电压时，不管有没有允许低压检测中断，LVDF/PCON.5都自动为1。**该位要**用软件清0，**清0后，如内部工作电压Vcc低于低压检测门槛电压，该位又被自动设置为1。**

在进入掉电工作状态前，如果低压检测电路未被允许可产生中断，则在进入掉电模式后，该低压检测电路不工作以降低功耗。如果被允许可产生低压检测中断(相应的中断允许位是ELVD/IE.6，中断请求标志位是LVDF/PCON.5)，则在进入掉电模式后，该低压检测电路继续工作，在内部工作电压Vcc**低于低压检测门槛电压后，产生低压检测中断，可将**MCU从掉电状态唤醒。

建议在电压偏低时，不要操作EEPROM/IAP, 烧录时直接选择“低压禁止EEPROM操作”。



与低压检测相关的一些寄存器：

PCON：电源控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF：低压检测标志位,同时也是低压检测中断请求标志位。

在正常工作和空闲工作状态时，如果内部工作电压Vcc低于低压检测门槛电压，该位自动置1，与低压检测中断是否被允许无关。即在内部工作电压Vcc低于低压检测门槛电压时，不管有没有允许低压检测中断，该位都自动为1。该位要用软件清0，清0后，如内部工作电压Vcc继续低于低压检测门槛电压，该位又被自动设置为1。

在进入掉电工作状态前，如果低压检测电路未被允许可产生中断，则在进入掉电模式后，该低压检测电路不工作以降低功耗。如果被允许可产生低压检测中断，则在进入掉电模式后，该低压检测电路继续工作，在内部工作电压Vcc低于低压检测门槛电压后，产生低压检测中断，可将MCU从掉电状态唤醒。

PD：　掉电模式控制位

IDL：　空闲模式控制位

GF1,GF0：两个通用工作标志位,用户可以任意使用。

STC MCU Limited

IE：中断允许寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：　中断允许总控制位
EA=0, 屏蔽所有的中断请求
EA=1, 开放中断,但每个中断源还有自己的独立允许控制位。

ELVD：低压检测中断允许位
ELVD = 0, 禁止低压检测中断
ELVD = 1, 允许低压检测中断

IP：中断优先级控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PLVD：低压检测中断优先级控制位
PLVD = 0, 低压检测中断位低优先级
PLVD = 1, 低压检测中断为高优先级

2.2.6 看门狗(WDT)复位

在工业控制/ 汽车电子/ 航空航天等需要高可靠性的系统中,为了防止“系统在异常情况下，受到干扰，MCU/CPU程序跑飞，导致系统长时间异常工作”,通常是引进看门狗,如果MCU/CPU 不在规定的时间内按要求访问看门狗,就认为MCU/CPU处于异常状态,看门狗就会强迫MCU/CPU复位，使系统重新从头开始按规律执行用户程序。

看门狗复位是热启动复位中的软复位之一。STC15系列单片机内部也引进了此看门狗功能， 使单片机系统可靠性设计变得更加方便/简洁。看门狗复位状态结束后，不影响特殊功能寄存器IAP_CONTR中SWBS/IAP_CONTR.6位的值，对于STC15F/L101W系列、STC15F/L2K60S2系列、STC15F/L408AD系列及STC15W408AS系列单片机，它们根据复位前SWBS/IAP_CONTR.6的值选择是从用户应用程序区启动，还是从系统ISP监控程序区启动。如果看门狗复位前它们的SWBS/IAP_CONTR.6的值为0，则看门口复位状态结束后上述系列单片机将从用户应用程序区启动。如果看门狗复位前它们的SWBS/IAP_CONTR.6的值为1，则看门口复位状态结束后上述系列单片机将从系统ISP监控程序区启动。对于STC15W201S系列、STC15W1K16S系列及STC15W408S系列单片机，它们的看门狗复位状态结束后始终从系统ISP监控程序区启动，与复位前SWBS/IAP_CONTR.6的值无关。

对于看门狗复位功能，我们增加如下特殊功能寄存器WDT_CONTR：

WDT_CONTR：看门狗(Watch-Dog-Timer)控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
WDT_CONTR	0C1H	name	WDT_FLAG	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0

Symbol符号　Function功能

WDT_FLAG :　When WDT overflows， this bit is set. It can be cleared by software.
看门狗溢出标志位,当溢出时，该位由硬件置1，可用软件将其清0。

EN_WDT :　Enable WDT bit. When set, WDT is started
看门狗允许位，当设置为“1 ”时，看门狗启动。

CLR_WDT :　WDT clear bit. If set, WDT will recount. Hardware will automatically clear this bit.
看门狗清“0”位,当设为“1”时,看门狗将重新计数。硬件将自动清“0 ”此位。

IDLE_WDT :　When set, WDT is enabled in IDLE mode. When clear, WDT is disabled in IDLE
看门狗“IDLE ”模式位，当设置为“1”时，看门狗定时器在“空闲模式”计数
当清“0”该位时，看门狗定时器在“空闲模式”时不计数

PS2,PS1,PS0 :　Pre-scale value of Watchdog timer is shown as the bellowed table:
看门狗定时器预分频值，如下表所示

The WDT period is determined by the following equation 看门狗溢出时间计算：
看门狗溢出时间 =（ 12 x Pre-scale x 32768) / Oscillator frequency

其中，设时钟为20MHz：
则：看门狗溢出时间 = (12 × Pre-scale × 32768) / 12000000 = Pre-scale× 393216 / 20000000

PS2	PS1	PS0	Pre-scale 预分频	WDT overflow Time @20MHz
0	0	0	2	39.3 mS
0	0	1	4	78.6 mS
0	1	0	8	157.3 mS
0	1	1	16	314.6 mS
1	0	0	32	629.1 mS
1	0	1	64	1.25 S
1	1	0	128	2.5 S
1	1	1	256	5 S

又设时钟为12MHz：

看门狗溢出时间 = (12 × Pre-scale × 32768) / 12000000 = Pre-scale× 393216 / 12000000

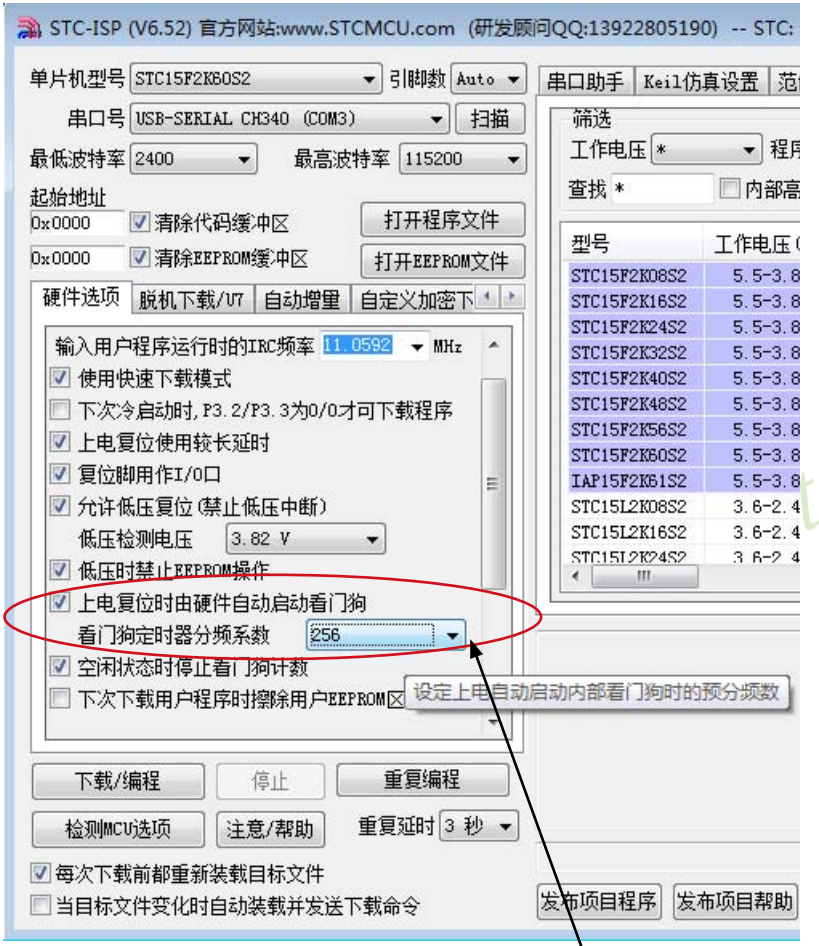
PS2	PS1	PS0	Pre-scale 预分频	WDT overflow Time @12MHz
0	0	0	2	65.5 mS
0	0	1	4	131.0 mS
0	1	0	8	262.1 mS
0	1	1	16	524.2 mS
1	0	0	32	1.0485 S
1	0	1	64	2.0971 S
1	1	0	128	4.1943 S
1	1	1	256	8.3886 S

再设时钟为11.0592MHz：

看门狗溢出时间= (12 x Pre-scale x 32768) / 11059200 = Pre-scale x 393216 / 11059200

PS2	PS1	PS0	Pre-scale	WDT overflow Time @11.0592MHz
0	0	0	2	71.1 mS
0	0	1	4	142.2 mS
0	1	0	8	284.4 mS
0	1	1	16	568.8 mS
1	0	0	32	1.1377 S
1	0	1	64	2.2755 S
1	1	0	128	4.5511 S
1	1	1	256	9.1022 S

STC MCU Limited



STC-ISP下编程器中看门狗的设置区

看门狗测试程序，在STC的下载板上可以直接测试

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ---------------------------------------*/
/* --- 演示STC 15 系列单片机 看门狗及其溢出时间计算公式--------*/
/* --- 研发顾问QQ：800003751----------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------*/
/*---- 在 Keil C 开发环境中，选择 Intel 8052 编译即可-------------------*/
/*---------------------------------------------------------------------------*/

;本演示程序在STC 15系列 ISP的下载编程工具上测试通过,相关的工作状态在P1口上显示
;看门狗及其溢出时间 = (12 * Pre_scale *32768)/Oscillator frequency
WDT_CONTR       EQU   0C1H ;看门狗地址
WDT_TIME_LED    EQU   P1.5 ;用 P1.5 控制看门狗溢出时间指示灯,
                           ;看门狗溢出时间可由该指示灯亮的时间长度或熄灭的时间长度表示
WDT_FLAG_LED    EQU   P1.7
                ;用 P1.7 控制看门狗溢出复位指示灯, 如点亮表示为看门狗溢出复位
Last_WDT_Time_LED_Status  EQU 00H ;位变量, 存储看门狗溢出时间指示灯的上一次状态位
;WDT 复位时间(所用的Oscillator frequency = 18.432MHz):
;Pre_scale_Word  EQU 00111100B    ;清0,启动看门狗, 预分频数=32,    0.68S
Pre_scale_Word  EQU 00111101B    ;清0,启动看门狗, 预分频数=64,    1.36S
;Pre_scale_Word  EQU 00111110B    ;清0,启动看门狗, 预分频数=128,   2.72S
;Pre_scale_Word  EQU 00111111B    ;清0,启动看门狗, 预分频数=256,   5.44S
    ORG   0000H
    AJMP  MAIN
    ORG   0100H
MAIN:
    MOV   A, WDT_CONTR          ;检测是否为看门狗复位
    ANL   A, #10000000B
    JNZ   WDT_Reset             ;WDT_CONTR.7 = 1, 看门狗复位, 跳转到看门狗复位程序
;WDT_CONTR.7 = 0,上电复位, 冷启动, RAM 单元内容为随机值
    SETB  Last_WDT_Time_LED_Status      ;上电复位,
                                        ;初始化看门狗溢出时间指示灯的状态位 = 1
    CLR   WDT_TIME_LED                  ;上电复位, 点亮看门狗溢出时间指示灯
    MOV   WDT_CONTR, #Pre_scale_Word    ;启动看门狗
WAIT1:
    SJMP  WAIT1                 ;循环执行本语句(停机), 等待看门狗溢出复位
;WDT_CONTR.7 = 1,看门狗复位, 热启动, RAM 单元内容不变, 为复位前的值
```

```
WDT_Reset:                          ;看门狗复位, 热启动
    CLR   WDT_FLAG_LED              ;是看门狗复位,点亮看门狗溢出复位指示灯

    JB   Last_WDT_Time_LED_Status, Power_Off_WDT_TIME_LED
                                    ;为1熄灭相应的灯,为0亮相应灯
    ;根据看门狗溢出时间指示灯的上一次状态位设置 WDT_TIME_LED 灯,
    ;若上次亮本次就熄灭, 若上次熄灭本次就亮
   CLR WDT_TIME_LED ;上次熄灭本次点亮看门狗溢出时间指示灯
   CPL Last_WDT_Time_LED_Status ;将看门狗溢出时间指示灯的上一次状态位取反
WAIT2:
   SJMP WAIT2 ;循环执行本语句(停机), 等待看门狗溢出复位
Power_Off_WDT_TIME_LED:
   SETB WDT_TIME_LED ;上次亮本次就熄灭看门狗溢出时间指示灯
   CPL Last_WDT_Time_LED_Status ;将看门狗溢出时间指示灯的上一次状态位取反
WAIT3:
   SJMP WAIT3          ;循环执行本语句(停机), 等待看门狗溢出复位
   END
```

STC MCU Limited

2.2.7 程序地址非法复位

如果程序指针PC指向的地址超过了有效程序空间的大小，就会引起程序地址非法复位。程序地址非法复位状态结束后，不影响特殊功能寄存器IAP_CONTR中SWBS/IAP_CONTR.6位的值，单片机将根据复位前SWBS/IAP_CONTR.6的值选择是从用户应用程序区启动，还是从系统ISP监控程序区启动。如果复位前SWBS/IAP_CONTR.6的值为0，则单片机从用户应用程序区启动。反之，如果复位前SWBS/IAP_CONTR.6的值为1，则单片机从系统ISP监控程序区启动。程序地址非法复位是热启动复位中的软复位之一。

2.2.8 热启动复位和冷启动复位

		复位源		现象	复位后SWBS/IAP_CONTR.6的值
热启动复位	软复位	软件复位 (通过控制IAP_CONTR特殊功能寄存器的其中两位SWBS/SWRST实现复位)	通过对IAP_CONTR寄存器送入20H产生的软复位	会使系统从用户应用程序区0000H处开始执行用户程序	0
			通过对IAP_CONTR寄存器送入60H产生的软复位	会使系统从系统ISP监控程序区开始执行程序，检测不到合法的ISP下载命令流后，或检测到合法的ISP下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1
		看门狗复位 (由MCU/CPU不在规定的时间内按要求访问看门狗所引起的复位) (不影响SWBS/IAP_CONTR.6的值)	复位前SWBS/IAP_CONTR.6的值为0	会使系统从用户应用程序区0000H处开始执行用户程序	0
			复位前SWBS/IAP_CONTR.6的值为1	会使系统从系统ISP监控程序区开始执行程序，检测不到合法的ISP下载命令流后，或检测到合法的ISP下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1
		程序地址非法复位 (由程序指针PC指向的地址超过有效程序空间的大小所引起的复位) (不影响SWBS/IAP_CONTR.6的值)	复位前SWBS/IAP_CONTR.6的值为0	会使系统从用户应用程序区0000H处开始执行用户程序	0
			复位前SWBS/IAP_CONTR.6的值为1	会使系统从系统ISP监控程序区开始执行程序，检测不到合法的ISP下载命令流后，或检测到合法的ISP下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1
	硬复位	内部低压检测复位 (当用户在ISP编程时允许低压检测复位并且电源电压Vcc在上电复位门槛电压以上时，由电源电压Vcc低于内部低压检测门槛电压所产生的复位) (不影响SWBS/IAP_CONTR.6的值)	复位前SWBS/IAP_CONTR.6的值为0	会使系统从用户应用程序区0000H处开始执行用户程序	0
			复位前SWBS/IAP_CONTR.6的值为1	会使系统从系统ISP监控程序区开始执行程序，检测不到合法的ISP下载命令流后，或检测到合法的ISP下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1
		外部RST引脚复位 (通过从外部向RST引脚施加一定宽度的复位脉冲所产生的复位)		会将特殊功能寄存器IAP_CONTR中的SWBS/IAP_CONTR.6位置1，同时会使系统从系统ISP监控程序区开始执行程序，检测不到合法的ISP下载命令流后，或检测到合法的ISP下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1
冷启动复位		冷启动复位即掉电复位/上电复位 系统停电后再上电引起的复位 当电源电压Vcc低于掉电复位检测门槛电压时，就会引起系统复位，此复位称为掉电复位，如果电源电压Vcc再次上升至上电复位检测门槛电压(与掉电复位检测门槛电压相等)以上时，系统仍处于复位状态，此时称为上电复位，上电复位延时32768个时钟后，复位状态才会结束。如果用户在ISP编程时选择了180mS的长复位延时，则上电复位后将产生约180mS复位延时，复位状态才结束。 (对于5V单片机，目前掉电复位/上电复位检测门槛电压约为3.2V；对于3.3V单片机，目前掉电复位/上电复位检测门槛电压约为1.8V)		会将特殊功能寄存器IAP_CONTR中的SWBS/IAP_CONTR.6位置1,同时会使系统从系统ISP监控程序区开始执行程序，检测不到合法的ISP下载命令流后，或检测到合法的ISP下载命令流并下载完用户程序后，均会软复位到用户应用程序区执行用户程序	1

IAP_CONTR: ISP/IAP 控制寄存器

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	name	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0

SWBS：软件选择复位后从用户应用程序区启动(送0)，还是从系统ISP监控程序区启动(送1)。要与SWRST直接配合才可以实现

SWRST：0：不操作； 1：软件控制产生复位，单片机自动复位。

2.3 STC15系列单片机的省电模式

STC15系列单片机可以运行3种省电模式以降低功耗，它们分别是：低速模式，空闲模式和掉电模式。正常工作模式下，STC15系列单片机的典型功耗是2.7mA ~ 7mA，而掉电模式下的典型功耗是<0.1uA，空闲模式下的典型功耗是1.8mA.

低速模式由时钟分频器CLK_DIV (PCON2)控制，而空闲模式和掉电模式的进入由电源控制寄存器PCON的相应位控制。PCON寄存器定义如下：

PCON (Power Control Register)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

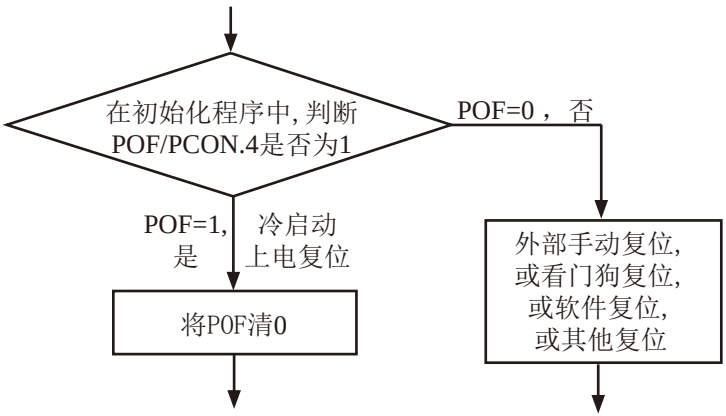
LVDF：低压检测标志位,同时也是低压检测中断请求标志位。

在正常工作和空闲工作状态时，如果内部工作电压Vcc低于低压检测门槛电压，该位自动置1，与低压检测中断是否被允许无关。即在内部工作电压Vcc低于低压检测门槛电压时，不管有没有允许低压检测中断，该位都自动为1。该位要用软件清0，清0后，如内部工作电压Vcc继续低于低压检测门槛电压，该位又被自动设置为1。

在进入掉电工作状态前，如果低压检测电路未被允许可产生中断，则在进入掉电模式后，该低压检测电路不工作以降低功耗。如果被允许可产生低压检测中断，则在进入掉电模式后，该低压检测电路继续工作，在内部工作电压Vcc低于低压检测门槛电压后，产生低压检测中断，可将MCU从掉电状态唤醒。

POF：　上电复位标志位，单片机停电后，上电复位标志位为1 ，可由软件清0 。

实际应用：要判断是上电复位（冷启动），还是外部复位脚输入复位信号产生的复位，还是内部看门狗复位，**还是软件复位或者其他复位，可通过如下方法来判断：**



判断复位种类流程图

PD：将其置1时，进入Power Down模式，可由外部中断上升沿触发或下降沿触发唤醒，进入掉电模式时，内部时钟停振，由于无时钟，所以CPU、定时器等功能部件停止工作，只有外部中断继续工作。可将CPU从掉电模式唤醒的外部管脚有：INT0/P3.2, INT1/P3.3, $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0；管脚CCP0/CCP1/CCP2；管脚RxD/RxD2/RxD3/RxD4；管脚T0/T1/T2/T3/T4；有些单片机还具有内部低功耗掉电唤醒专用定时器。掉电模式也叫停机模式，此时功耗<0.1uA。

IDL：将其置1，进入IDLE模式(空闲)，除系统不给CPU供时钟，CPU不执行指令外，其余功能部件仍可继续工作，可由外部中断、定时器中断、低压检测中断及A/D转换中断中的任何一个中断唤醒。

GF1,GF0：两个通用工作标志位,用户可以任意使用。

SMOD, SMOD0：与电源控制无关，与串口有关，在此不作介绍。

STC MCU Limited

2.3.1 低速模式及其测试程序(C和汇编)

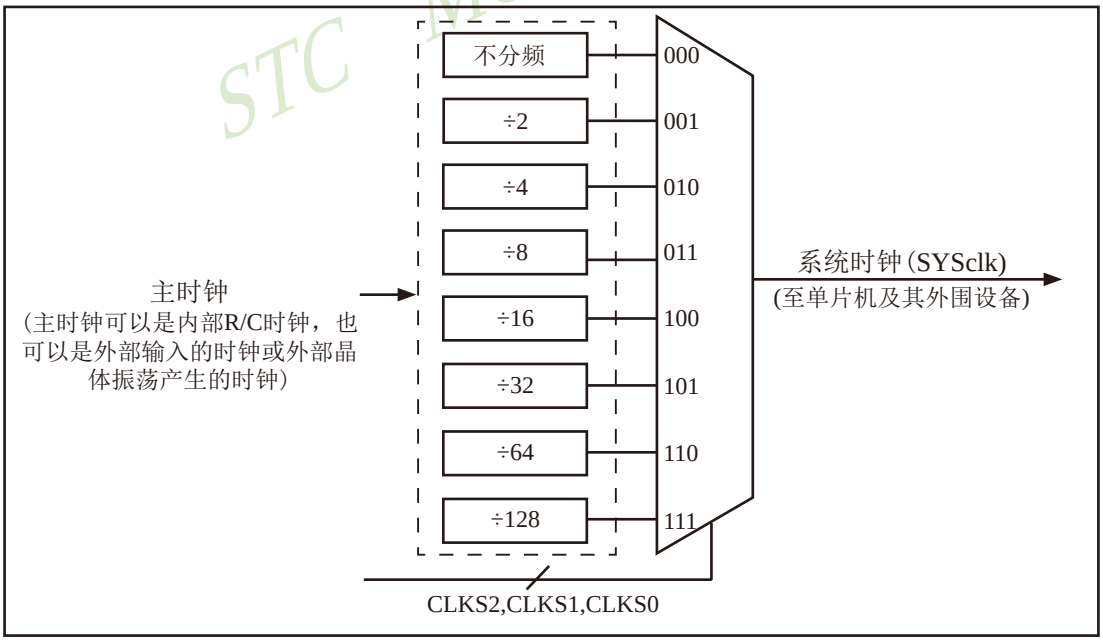
时钟分频器可以对内部时钟进行分频，从而降低工作时钟频率，降低功耗，降低EMI。

时钟分频寄存器CLK_DIV (PCON2)各位的定义如下：

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 (系统时钟是指对主时钟进行分频后供给CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D转换的实际工作时钟)
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。



时钟结构

1.C程序

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 低速模式举例--------------------------------------------------------*/
/* --- 研发顾问QQ：  800003751---------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

sfr         CLK_DIV     =  0x97;          //时钟分频寄存器

//-----------------------------------------

void main()
{
            CLK_DIV = 0x00;               //系统时钟为主时钟
//          CLK_DIV = 0x01;               //系统时钟为主时钟/2
//          CLK_DIV = 0x02;               //系统时钟为主时钟/4
//          CLK_DIV = 0x03;               //系统时钟为主时钟/8
//          CLK_DIV = 0x04;               //系统时钟为主时钟/16
//          CLK_DIV = 0x05;               //系统时钟为主时钟/32
//          CLK_DIV = 0x06;               //系统时钟为主时钟/64
//          CLK_DIV = 0x07;               //系统时钟为主时钟/128

            while (1);                    //程序终止
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 低速模式举例--------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

CLK_DIV   DATA        097H                  //时钟分频寄存器

//-----------------------------------------
          ORG     0000H
          LJMP    MAIN                    //复位入口
//-----------------------------------------

          ORG     0100H
MAIN:
          MOV     SP,     #3FH

          MOV     CLK_DIV,    #0          //系统时钟为主时钟
//        MOV     CLK_DIV,    #1          //系统时钟为主时钟/2
//        MOV     CLK_DIV,    #2          //系统时钟为主时钟/4
//        MOV     CLK_DIV,    #3          //系统时钟为主时钟/8
//        MOV     CLK_DIV,    #4          //系统时钟为主时钟/16
//        MOV     CLK_DIV,    #5          //系统时钟为主时钟/32
//        MOV     CLK_DIV,    #6          //系统时钟为主时钟/64
//        MOV     CLK_DIV,    #7          //系统时钟为主时钟/128

          SJMP    $                       //程序终止

;---------------------------------------------

          END
```

STC MCU Limited

2.3.2 空闲模式(功耗<1mA)及其测试程序(C和汇编)

将IDL/PCON.0置为1，单片机将进入IDLE(空闲)模式。在空闲模式下，仅CPU无时钟停止工作，但是外部中断、内部低压检测电路、定时器、A/D转换等仍正常运行。而看门狗在空闲模式下是否工作取决于其自身有一个“IDLE ”模式位：IDLE_WDT(WDT_CONTR.3)。当IDLE_WDT位被设置为“1”时， 看门狗定时器在“空闲模式”计数，即正常工作。当IDLE_WDT位被清“0”时， 看门狗定时器在“空闲模式”时不计数，即停止工作。在空闲模式下，RAM、堆栈指针(SP)、程序计数器(PC)、程序状态字(PSW)、累加器(A)等寄存器都保持原有数据。I/O口保持着空闲模式被激活前那一刻的逻辑状态。空闲模式下单片机的所有外围设备都能正常运行(除CPU无时钟不工作外)。当任何一个中断产生时，它们都可以将单片机唤醒，单片机被唤醒后，CPU将继续执行进入空闲模式语句的下一条指令。

有两种方式可以退出空闲模式。任何一个中断的产生都会引起IDL/PCON.0被硬件清除，从而退出空闲模式。另一个退出空闲模式的方法是：外部RST引脚复位，将复位脚拉高，产生复位。这种拉高复位引脚来产生复位的信号源需要被保持24个时钟加上20us，才能产生复位，再将RST引脚拉低，结束复位，单片机从系统ISP监控程序区开始启动。

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 空闲模式举例--------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/
//特别注意：在将进入空闲模式时一定要加入1-4条_nop_()语句（空语句），即一定要在设置MCU进入
//空闲模式的语句后加1-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz
#include "reg51.h"
#include "intrins.h"

//-----------------------------------------
void main()
{
        while (1)
        {
                PCON |= 0x01;                   //将IDL(PCON.0)置1,MCU将进入空闲模式
                _nop_();                        //此时CPU无时钟,不执行指令
                _nop_();                        //内部中断信号和外部复位信号可以终止空闲模式
                _nop_();
                _nop_();
        }
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 空闲模式举例--------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//特别注意：在将进入空闲模式时一定要加入1-4条NOP语句（空语句），即一定要在设置MCU进入
//空闲模式的语句后加1-4条NOP语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

//-----------------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

LOOP:
        MOV     PCON,   #01H            //将IDL(PCON.0)置1,MCU将进入空闲模式
        NOP                             //此时CPU无时钟,不执行指令
        NOP                             //内部中断信号和外部复位信号可以终止空闲模式
        NOP
        NOP
        JMP     LOOP

;-----------------------------------------------

        END
```

STC MCU Limited

2.3.3 掉电模式/停机模式及其测试程序(C和汇编)

将PD/PCON.1置为1，单片机将进入Power Down(掉电)模式，掉电模式也叫停机模式。进入掉电模式/停机模式后，单片机所使用的时钟(内部系统时钟或外部晶体/时钟)停振，由于无时钟源，CPU、看门狗、定时器、串行口、A/D转换等功能模块停止工作，外部中断(INT0/INT1/$\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$)、CCP继续工作。如果低压检测电路被允许可产生中断，则低压检测电路也可继续工作，否则将停止工作。进入掉电模式/停机模式后，所有I/O口、SFRs(特殊功能寄存器)维持进入掉电模式/停机模式前那一刻的状态不变。如果掉电唤醒专用定时器在进入掉电模式之前被打开(即在进入掉电模式/停机模式之前WKTEN/WKTCH.7=1)，则进入掉电模式/停机模式后，掉电唤醒专用定时器将开始工作。

进入掉电模式/停机模式后，STC15W4K60S4系列单片机中可将掉电模式/停机模式唤醒的管脚资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚CCP0/CCP1/CCP2；管脚RxD/RxD2/RxD3/RxD4；管脚T0/T1/T2/T3/T4(下降沿即外部管脚T0/T1/T2/T3/T4由高到低的变化，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；低压检测中断(前提是低压检测中断被允许即ELVD/IE.6被置1,且在STC-ISP编程/烧录用户应用程序时不选择“允许低压复位/禁止低压中断”)；内部低功耗掉电唤醒专用定时器。

STC15系列单片机的内部低功耗掉电唤醒专用定时器由特殊功能寄存器WKTCH和WKTCL进行管理和控制。

WKTCL(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCL	AAH	name									1111 11110B

WKTCH(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCH	ABH	name	WKTEN								0111 1111B

内部掉电唤醒定时器是一个15位定时器，{WKTCH[6:0],WKTCL[7:0]}构成最长15位计数值(32768个),定时从0开始计数。

WKTEN：内部停机唤醒定时器的使能控制位。

WKTEN=1，允许内部停机唤醒定时器；

WKTEN=0，禁止内部停机唤醒定时器；

STC15系列有内部低功耗掉电唤醒专用定时器的单片机除增加了特殊功能寄存器WKTCL和WKTCH，还设计了了2个隐藏的特殊功能寄存器WKTCL_CNT和WKTCH_CNT来控制内部掉电唤醒专用定时器。WKTCL_CNT与WKTCL共用同一个地址，WKTCH_CNT与WKTCH共用同一个地址，WKTCL_CNT和WKTCH_CNT是隐藏的，对用户不可见。WKTCL_CNT和WKTCH_CNT实际上是作计数器使用，而WKTCL和WKTCH实际上作比较器使用。当用户对WKTCL和WKTCH写入内容时，该内容只写入寄存器WKTCL和WKTCH中，而不会写入WKTCL_CNT和WKTCH_CNT中。当用户读寄存器WKTCL和WKTCH中的内容时，实际上读的是寄存器WKTCL_CNT和WKTCH_CNT中的内容，而不是WKTCL和WKTCH中的内容。

STC MCU Limited

特殊功能寄存器WKTCL_CNT和WKTCH_CNT的格式如下所示：

WKTCL_CNT

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCL_CNT	AAH	name									1111 1111B

WKTCH_CNT

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCH_CNT	ABH	name	-								x111 1111B

如果STC15系列单片机内置掉电唤醒专用定时器被允许(通过软件将WKTCH寄存器中的WKTEN/WKTCH.7位置‘1’,就可以打开内部掉电唤醒专用定时器)，当MCU进入掉电模式/停机模式后，掉电唤醒专用定时器开始工作，MCU可由该掉电唤醒专用定时器唤醒。掉电唤醒专用定时器将MCU从掉电模式/停机模式唤醒的执行过程是：一旦MCU进入掉电模式/停机模式,内部掉电唤醒专用定时器[WKTCH_CNT,WKTCL_CNT]就从7FFFH开始计数,直到计数到与{WKTCH[6:0],WKTCL[7:0]}寄存器所设定的计数值相等后就让系统时钟开始振荡；如果主时钟使用的是内部系统时钟(由用户在ISP烧录程序时自行设置)，MCU在等待64个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU、定时器、看门狗、A/D转换等功能模块工作；如果主时钟使用的是外部晶体或时钟(由用户在ISP烧录程序时自行设置)，MCU在等待1024个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU、定时器、看门狗、A/D转换等功能模块工作；CPU获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。掉电唤醒之后，WKTCH_CNT和WKTCL_CNT的内容保持不变，因此可以通过读[WKTCH,WKTCL]的内容（实际上是读[WKTCH_CNT,WKTCL_CNT]的内容）读出单片机在停机模式/掉电模式所等待的时间。

这里请注意：用户在设置寄存器{WKTCH[6:0],WKTCL[7:0]}的计数值时，要按照所需要的计数次数，在计数次数的基础上减1所得的数值才是{WKTCH,WKTCL}的计数值。如用户需计数10次，则将9写入寄存器{WKTCH[6:0],WKTCL[7:0]}中。同样，如果用户需计数32768次，则应对{WKTCH[6:0],WKTCL[7:0]}写入7FFFH(即32767)。

内部掉电唤醒定时器有自己的内部时钟，其中掉电唤醒定时器计数一次的时间就是由该时钟决定的。内部掉电唤醒定时器的时钟频率约为32768Hz，当然误差较大。对于16-pin及其以上的单片机，用户可以通过读RAM区F8单元和F9单元的内容来获取内部掉电唤醒专用定时器常温下的时钟频率。对于8-pin单片机即STC15F101W系列，用户可以通过读RAM区78单元和79单元的内容来获取内部掉电唤醒专用定时器常温下的时钟频率。下面以16-pin及其以上的单片机为例，介绍如何计算内部掉电唤醒专用定时器的计数时间。

假设我们用[WIRC_H,WIRC_L]来表示从RAM区F8单元和F9单元获取到的内部掉电唤醒专用定时器常温下的时钟频率，则内部掉电唤醒专用定时器计数时间按下式计算：

$$\text{内部掉电唤醒专用定时器计数时间} = \frac{10^6\,\text{uS}}{[\text{WIRC\_H, WIRC\_L}]} \times 16 \times \text{计数次数}$$

例如：假设读到RAM区F8单元的内容为80H，F9单元的内容为00H，即内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为32768Hz，则内部掉电唤醒专用定时器最短计数时间(即计数一次的时间)为：　$\frac{10^6\,uS}{32768}$ x 16 x 1 ≈ 488.28 uS

内部掉电唤醒专用定时器最长计数时间约为 488.28us x 32768 = 16S

设定{WKTCH[6:0],WKTCL[7:0]}寄存器的值等于9(即计数10次)且内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为32768Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 488.28uS x 10 ≈ 4882.8uS

设定{WKTCH[6:0],WKTCL[7:0]}寄存器的值等于32767(即最大计数值 = 32768 = 2^{15})且内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为32768Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 488.28uS x 32768 = 16S

下面给出了在读到RAM区F8单元的内容为80H，F9单元的内容为00H，即内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为32768Hz情况下，内部掉电唤醒专用定时器的计数时间：

{WKTCH[6:0],WKTCL[7:0]} = 0，	488.28uS x 1	= 488.28uS
{WKTCH[6:0],WKTCL[7:0]} = 9，	488.28uS x 10	= 4.8828mS
{WKTCH[6:0],WKTCL[7:0]} = 99，	488.28uS x 100	= 48.828mS
{WKTCH[6:0],WKTCL[7:0]} = 999，	488.28uS x 1000	= 488.28mS
{WKTCH[6:0],WKTCL[7:0]} = 4095，	488.28uS x 4096	= 2.0S
{WKTCH[6:0],WKTCL[7:0]} = 32767，	488.28uS x 32768=16S	

除掉电唤醒专用定时器外，还可将掉电模式/停机模式唤醒的中断有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0 ($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚CCP(CCP可以在CCP0/P1.1, CCP1/P1.0, CCP2/CCP2_2/P3.7, CCP0_2/P3.5, CCP1_2/P3.6, CCP2/CCP2_2/P3.7, CCP0_3/P2.5, CCP1_3/P2.6, CCP2_3/P2.7之间切换)。如果掉电模式/停机模式是由外部中断INT0(上升沿+下降沿中断)、INT1(上升沿+下降沿中断)、$\overline{\text{INT2}}$(仅可下降沿中断)、$\overline{\text{INT3}}$(仅可下降沿中断)、$\overline{\text{INT4}}$(仅可下降沿中断)或CCP管脚唤醒，则掉电唤醒之后CPU首先执行设置单片机进入掉电模式的语句的下一条语句（建议在设置单片机进入掉电模式的语句后多加几个NOP空指令），然后执行相应的中断服务程序。

另外，在串行中断被允许后，串行口1、串行口2、串行口3和串行口4的接收管脚RxD(可以在RxD/P3.0, RxD_2/P3.6, RxD_3/P2.6之间切换)、RxD2(可以在RxD2/P1.0, RxD2_2/P4.6之间切换)、RxD3(可以在RxD3/P0.0, RxD3_2/P5.0之间切换)和RxD4(可以在RxD4/P0.2, RxD4_2/P5.2之间切换)如发生由高到低的变化时(起始位接收)也可以将MCU从掉电模式/停机模式唤醒。当MCU

STC MCU Limited

由RxD或RxD2或RxD3或RxD4唤醒时，如果主时钟使用的是内部系统时钟(由用户在ISP烧录程序时自行设置)，MCU在等待64个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU工作；如果主时钟使用的是外部晶体或时钟(由用户在ISP烧录程序时自行设置)，MCU在等待1024个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU工作；CPU获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。

如果定时器T0/T1/T2/T3/T4的中断在进入掉电模式/停机模式前被允许了，即进入掉电模式/停机模式前ET0/ET1/ET2/ET3/ET4及EA已经被设置为1，则进入掉电模式/停机模式后，定时器T0/T1/T2/T3/T4的外部管脚(T0/P3.4, T1/P3.5, T2/P3.1, T3/P0.5, T4/P0.7)如发生由高到低的变化可以将MCU从掉电模式/停机模式唤醒。当MCU由定时器T0/T1/T2/T3/T4的外部管脚由高到低的变化唤醒时，如果主时钟使用的是内部系统时钟(由用户在ISP烧录程序时自行设置)，MCU在等待64个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU工作；如果主时钟使用的是外部晶体或时钟(由用户在ISP烧录程序时自行设置)，MCU在等待1024个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU工作；CPU获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行，不进入相应定时器的中断程序。

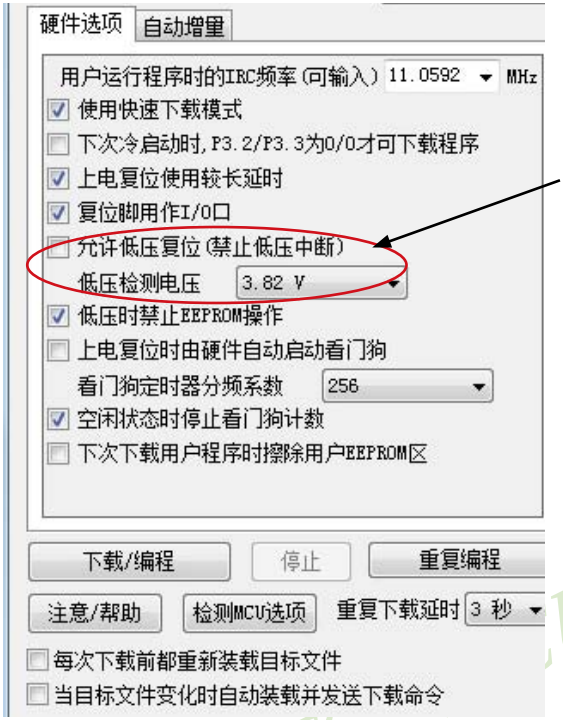
还有外部RST引脚复位也可将MCU从掉电模式/停机模式中唤醒，复位唤醒后的MCU将从系统ISP监控程序区开始启动。

特别声明：以15L开头的芯片如需进入"掉电模式",进入"掉电模式"前必须启动掉电唤醒定时器<3uA>,不超过1秒要唤醒一次，以15F和15W开头的芯片以及新供货的STC15L2K60S2系列D版本芯片则不需要

如果在STC-ISP编程/烧录用户应用程序时，不将低压检测设置为低压检测复位，则在用户程序中用户可将低压检测设置为低压检测中断。当电源电压VCC低于内部低压检测(LVD)门槛电压时，低压检测中断请求标志位(LVDF/PCON.5)就会被硬件置位。如果ELVD/IE.6(低压检测中断允许位)被设置为1，低压检测中断请求标志位就能产生一个低压检测中断。

在进入掉电模式/停机模式前，如果低压检测电路未被允许可产生中断，则在进入掉电模式/停机模式后，该低压检测电路不工作以降低功耗。如果被允许可产生低压检测中断(相应的中断允许位是ELVD/IE.6，中断请求标志位是LVDF/PCON.5)，则在进入掉电模式/停机模式后，该低压检测电路继续工作，在内部工作电压Vcc低于低压检测门槛电压后，产生低压检测中断，可将MCU从掉电模式/停机模式唤醒。

建议在电压偏低时，不要操作EEPROM/IAP,烧录时直接选择“低压禁止EEPROM操作”。



在进入掉电模式/停机模式前，如果低压检测中断被允许(ELVD=1)，进入掉电模式/停机模式后产生的低压检测中断可将MCU从掉电模式/停机模式唤醒。

STC MCU Limited

注意：

现供货的STC15F2K60S2系列C版本单片机的RxD管脚和RxD2管脚暂时不能将掉电模式/停机模式唤醒！STC15F2K60S2及STC15L2K60S2系列下一升级版本——STC15W2K60S2系列单片机将会设计实现该计划功能，STC15W2K60S2系列单片机的RxD管脚和RxD2管脚将均可用于唤醒掉电模式/停机模式。同时，STC15W2K60S2系列单片机还将增加比较器的功能。

现供货的STC15F408AD系列C版本单片机的RxD管脚暂时也不能将掉电模式/停机模式唤醒！STC15F408AD及STC15L408AD系列下一升级版本——STC15W408AS系列单片机将会设计实现该计划功能，STC15W408AS系列单片机的RxD管脚将可用于唤醒掉电模式/停机模式。

2.3.3.1 掉电模式/停机模式被唤醒后程序执行流程说明及测试程序(C和汇编)

当STC15W4K60S4系列单片机内置掉电唤醒专用定时器被允许(WKTEN=1)，掉电唤醒专用定时器将MCU从掉电模式/停机模式唤醒的执行过程是：一旦MCU进入掉电模式/停机模式,内部掉电唤醒专用定时器[WKTCH_CNT,WKTCL_CNT]就从7FFFH开始计数,直到计数到与{WKTCH[6:0],WKTCL[7:0]}寄存器所设定的计数值相等后就让系统时钟开始振荡；如果主时钟使用的是内部系统时钟(由用户在ISP烧录程序时自行设置)，MCU在等待64个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU工作；如果主时钟使用的是外部晶体或时钟(由用户在ISP烧录程序时自行设置)，MCU在等待1024个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU工作；CPU获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。

掉电模式/停机模式由中断INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0 ($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚CCP(可以在CCP0/P1.1, CCP1/P1.0, CCP2/CCP2_2/P3.7, CCP0_2/P3.5, CCP1_2/P3.6, CCP2/CCP2_2/P3.7, CCP0_3/P2.5, CCP1_3/P2.6, CCP2_3/P2.7之间切换)唤醒之后程序的执行流程为：CPU首先执行从上次设置单片机进入掉电模式语句的下一条语句（建议在设置单片机进入掉电模式的语句后多加几个NOP空指令），然后执行相应的中断服务程序。

掉电模式/停机模式由串行口1、串行口2、串行口3和串行口4的接收管脚RxD(可以在RxD/P3.0, RxD_2/P3.6, RxD_3/P2.6之间切换)、RxD2(可以在RxD2/P1.0, RxD2_2/P4.6之间切换)、RxD3(可以在RxD3/P0.0, RxD3_2/P5.0之间切换)和RxD4(可以在RxD4/P0.2, RxD4_2/P5.2之间切换)的下降沿(不产生中断)唤醒后的程序执行流程：当MCU由RxD的下降沿或RxD2的下降沿或RxD3的下降沿或RxD4的下降沿唤醒后，如果主时钟使用的是内部系统时钟，MCU在等待64个时钟(由用户在ISP烧录程序时自行设置)后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,才将时钟供给CPU工作；如果主时钟使用的是外部晶体或时钟，MCU在等待1024个时钟(由用户在ISP烧录程序时自行设置)后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,才将时钟供给CPU工作；CPU获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。

掉电模式/停机模式由定时器T0/T1/T2/T3/T4的外部管脚的下降沿(不产生中断)唤醒后的程序执行流程：当MCU由定时器T0/T1/T2/T3/T4的外部管脚的下降沿唤醒后，如果主时钟使用的是内部系统时钟，MCU在等待64个时钟(由用户在ISP烧录程序时自行设置)后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,才将时钟供给CPU工作；如果主时钟使用的是外部晶体或时钟，MCU在等待1024个时钟(由用户在ISP烧录程序时自行设置)后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,才将时钟供给CPU工作；CPU获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行，不进入相应定时器的中断程序。

STC MCU Limited

1. C程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列  掉电模式中指令执行流程说明-----------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条_nop_()语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------

void main()
{
        while (1)
        {
                PCON |= 0x02;                   //将STOP(PCON.1)置1,MCU将进入掉电模式
                _nop_();
                //当有有效的掉电唤醒源产生时，若使用的是内部振荡器,则立即启动内部振荡器，
                //在64个时钟周期后，将时钟提供给MCU,作为系统时钟；若使用的是外部振荡器，
                //则立即启动外部振荡器,在1024个时钟周期后,将时钟提供给MCU,作为系统时钟。
                //在时钟信号到达CPU后，若掉电唤醒源是内部32K掉电唤醒定时器、RxD和RxD2时，
                //CPU直接从此语句开始向下执行程序代码，而不产生中断；若掉电唤醒源是INT0、
                //INT1、INT2、INT3、INT4、CCP0、CCP1、CCP2时，则CPU首先执行此语句，
                //然后执行中断服务程序。

                _nop_();
                _nop_();
                _nop_();
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 掉电模式中指令执行流程说明-----------------------------*/
/* --- 研发顾问QQ：800003751---------------- ---------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,在程序中注明使用了STC的资料及程序 */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条NOP语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条NOP语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz
//-----------------------------------------------
        ORG     0000H
        LJMP    MAIN                    //复位入口

//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

LOOP:
        MOV     PCON,   #02H            //将STOP(PCON.1)置1,MCU将进入掉电模式

        NOP     //当有有效的掉电唤醒源产生时，若使用的是内部振荡器,则立即启动内部振荡器，
                //在64个时钟周期后，将时钟提供给MCU,作为系统时钟；若使用的是外部振荡器，
                //则立即启动外部振荡器,在1024个时钟周期后,将时钟提供给MCU,作为系统时钟。
                //在时钟信号到达CPU后，若掉电唤醒源是内部32K掉电唤醒定时器、RxD和RxD2时，
                //CPU直接从此语句开始向下执行程序代码，而不产生中断；若掉电唤醒源是INT0、
                //INT1、INT2、INT3、INT4、CCP0、CCP1、CCP2时，则CPU首先执行此语句，
                //然后执行中断服务程序。

        NOP
        NOP
        NOP
        JMP     LOOP

;-----------------------------------------------

        END
```

STC MCU Limited

2.3.3.2 用掉电唤醒专用定时器唤醒掉电模式/停机模式的测试程序(C和汇编)

——以15L开头的单片机进入掉电模式/停机模式前必须启动掉电唤醒专用定时器

特别声明：以15L开头的芯片如需进入"掉电模式",进入"掉电模式"前必须启动掉电唤醒定时器<3uA>,不超过1秒要唤醒一次，以15F和15W开头的芯片以及新供货的STC15L2K60S2系列D版本芯片则不需要

/*利用内部专用掉电唤醒定时器来唤醒掉电模式的示例程序（C程序）

1. C程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/* --- 演示STC15W4K60S4 系列 掉电唤醒定时器举例----------------------------------------------------*/
/* --- 研发顾问QQ：  800003751----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------- */

//特别注意：在将进入掉电模式时一定要加入2-4条_nop_()语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------
sfr     WKTCL  =        0xaa;           //掉电唤醒定时器计时低字节
sfr     WKTCH =         0xab;           //掉电唤醒定时器计时高字节

sbit    P10 = P1^0;
//-----------------------------------------
void main()
{
        WKTCL = 49;                     //设置唤醒周期为488us*(49+1) = 24.4ms
        WKTCH = 0x80;                   //使能掉电唤醒定时器

        while (1)
        {
                PCON = 0x02;            //进入掉电模式
                _nop_();                //掉电模式被唤醒后,直接从此语句开始向下执行,
                                        //不进入中断服务程序
                _nop_();

                P10 = !P10;             //掉电唤醒后,取反测试口
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/* --- 演示STC15W4K60S4 系列 掉电唤醒定时器举例----------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------                      */
/*---------------------------------------------------------------------------------------------------- */

//特别注意：在将进入掉电模式时一定要加入2-4条NOP语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条NOP语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

WKTCL  DATA   0AAH                    //掉电唤醒定时器计时低字节
WKTCH  DATA   0ABH                    //掉电唤醒定时器计时高字节

//---------------------------------------

        ORG     0000H
        LJMP    MAIN                  //复位入口

//---------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        MOV     WKTCL, #49            //设置唤醒周期为488us*(49+1) = 24.4ms
        MOV     WKTCH,#80H            //使能掉电唤醒定时器
LOOP:
        MOV     PCON,   #02H          //进入掉电模式
        NOP                           //掉电模式被唤醒后, 直接从此语句开始向下执行,
                                      //不进入中断服务程序
        NOP
        CPL     P1.0                  //掉电唤醒后,取反测试口
        JMP     LOOP

        SJMP    $

;---------------------------------------

        END
```

STC MCU Limited

2.3.3.3 用外部中断INT0(上升沿+下降沿)唤醒掉电模式/停机模式的测试程序(C和汇编)

1. C程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 INT0唤醒掉电模式举例----------------------------------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/
//特别注意：在将进入掉电模式时一定要加入2-4条_nop_()语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------
bit     FLAG;                   //1:上升沿中断 0:下降沿中断
sbit    P10     =       P1^0;

//-----------------------------------------
//中断服务程序
void exint0() interrupt 0               //INT0中断入口
{
        P10     =       !P10;           //将测试口取反
        FLAG    =       INT0;           //保存INT0口的状态, INT0=0(下降沿); INT0=1(上升沿)
}
//-----------------------------------------

void main()
{
        IT0 = 0;                        //设置INT0的中断类型为上升沿和下降沿,上升沿和下降沿均可唤醒
//      IT0 = 1;                        //设置INT0的中断类型为仅下降沿,下降沿唤醒

        EX0 = 1;                        //使能INT0中断
        EA = 1;

        while (1)
        {
                PCON = 0x02;            //MCU进入掉电模式
                _nop_();                //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
                _nop_();
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 INT0唤醒掉电模式举例----------------------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条NOP语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条NOP语句（空语句），如本程序中所示。
//假定测试芯片的工作频率为18.432MHz

FLAG    BIT     20H.0           //1:上升沿中断 0:下降沿中断
//---------------------------------------
        ORG     0000H
        LJMP    MAIN            //复位入口

        ORG     0003H           //INT0中断入口
        LJMP    EXINT0
//---------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        CLR     IT0             //设置INT0的中断类型为上升沿和下降沿,上升沿和下降沿均可唤醒
//      SETB    IT0             //设置INT0的中断类型为仅下降沿,下降沿唤醒

        SETB    EX0             //使能INT0中断
        SETB    EA

LOOP:
        MOV     PCON,   #02H    //MCU进入掉电模式
        NOP                     //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        NOP
        SJMP    LOOP
//---------------------------------------
//中断服务程序

EXINT0:
        CPL     P1.0            //将测试口取反
        PUSH    PSW
        MOV     C,      INT0    //读取INT0口的状态
        MOV     FLAG,   C       //保存, INT0=0(下降沿); INT0=1(上升沿)
        POP     PSW
        RETI
;---------------------------------------
        END
```

STC MCU Limited

2.3.3.4 用外部中断INT1(上升沿+下降沿)唤醒掉电模式/停机模式的测试程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 INT1唤醒掉电模式举例----------------------------------------*/
/* --- 研发顾问QQ：800003751---------------- ----------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条_nop_()语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------
bit     FLAG;                           //1:上升沿中断 0:下降沿中断
sbit    P10     =       P1^0;
//-----------------------------------------
//中断服务程序
void exint1() interrupt 2                       //INT1中断入口
{
        P10     =       !P10;           //将测试口取反
        FLAG    =       INT1;           //保存INT1口的状态, INT1=0(下降沿); INT1=1(上升沿)
}
//-----------------------------------------

void main()
{
        IT1     =       0;      //设置INT1的中断类型为上升沿和下降沿,上升沿和下降沿均可唤醒
//      IT1     =       1;      //设置INT1的中断类型为仅下降沿,下降沿唤醒

        EX1     =       1;      //使能INT1中断
        EA      =       1;

        while (1)
        {
                PCON = 0x02;    //MCU进入掉电模式
                _nop_();        //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
                _nop_();
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 INT1唤醒掉电模式举例----------------------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条NOP语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条NOP语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

FLAG    BIT     20H.0                   //1:上升沿中断 0:下降沿中断
//---------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     0013H                   //INT1中断入口
        LJMP    EXINT1
//---------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        CLR     IT1             //设置INT1的中断类型为上升沿和下降沿,上升沿和下降沿均可唤醒
//      SETB    IT1             //设置INT1的中断类型为仅下降沿,下降沿唤醒

        SETB    EX1             //使能INT1中断
        SETB    EA

LOOP:
        MOV     PCON,   #02H    //MCU进入掉电模式
        NOP                     //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        NOP
        SJMP    LOOP
;---------------------------------------
EXINT1:
        CPL     P1.0                    //取反测试口
        PUSH    PSW
        MOV     C,      INT1            //读取INT1口的状态
        MOV     FLAG,   C               //保存, INT1=0(下降沿); INT0=1(上升沿)
        POP     PSW
        RETI
;---------------------------------------
        END
```

STC MCU Limited

2.3.3.5 用外部中断$\overline{\text{INT2}}$(下降沿)唤醒掉电模式/停机模式的测试程序(C和汇编)

1. C程序：

```
/*-----------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 INT2下降沿唤醒掉电模式举例-----------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*-----------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条_nop_()语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------------

sfr     INT_CLKO    =    0x8F;          //外部中断与时钟输出控制寄存器
sbit    INT2    =    P3^6;              //INT2引脚定义

sbit    P10     =    P1^0;
//-----------------------------------------
//中断服务程序
void exint2() interrupt 10
{
        P10     =    !P10;              //将测试口取反

//      INT_CLKO  &=  0xEF;             //若需要手动清除中断标志,可先关闭中断,
                                        //此时系统会自动 清除内部的中断标志
//      INT_CLKO   |=  0x10;            //然后再开中断即可
}
//-----------------------------------------------

void main()
{
        INT_CLKO  |=  0x10;             //(EX2 = 1)使能INT2下降沿中断
        EA = 1;

        while (1)
        {
                PCON = 0x02;            //MCU进入掉电模式
                _nop_();        //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
                _nop_();
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 INT2下降沿唤醒掉电模式举例----------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条NOP语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条NOP语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

INT_CLKO        DATA    08FH            //外部中断与时钟输出控制寄存器
INT2            BIT     P3.6            //INT2引脚定义
//---------------------------------------
        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     0053H                   //INT2中断入口
        LJMP    EXINT2
//---------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     INT_CLKO,       #10H    //(EX2 = 1)使能INT2下降沿中断

        SETB    EA

LOOP:
        MOV     PCON,   #02H            //MCU进入掉电模式
        NOP                             //掉电模式被唤醒后,首先执行此语句,
                                        //然后再进入中断服务程序
        NOP
        SJMP    LOOP
//---------------------------------------
//中断服务程序
EXINT2:
        CPL     P1.0                    //将测试口取反

//      ANL     INT_CLKO,       #0EFH   //若需要手动清除中断标志,可先关闭中断,
                                        //此时系统会自动清除内部的中断标志
//      ORL     INT_CLKO,       #10H    //然后再开中断即可

        RETI
;---------------------------------------
        END
```

STC MCU Limited

2.3.3.6 用外部中断INT3(下降沿)唤醒掉电模式/停机模式的测试程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 INT3下降沿唤醒掉电模式举例----------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条_nop_()语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----------------------------------------
sfr     INT_CLKO    =   0x8F;       //外部中断与时钟输出控制寄存器
sbit    INT3    =   P3^7;           //INT3引脚定义

sbit    P10     =   P1^0;

//-----------------------------------------
//中断服务程序
void exint3() interrupt 11
{
        P10     =   !P10;           //将测试口取反

//      INT_CLKO    &=  0xDF;       //若需要手动清除中断标志,可先关闭中断,
                                    //此时系统会自动清除内部的中断标志
//      INT_CLKO    |=  0x20;       //然后再开中断即可
}
//-----------------------------------------

void main()
{
        INT_CLKO    |=  0x20;       //(EX3 = 1)使能INT3下降沿中断
        EA          =   1;

        while (1)
        {
                PCON = 0x02;                //MCU进入掉电模式
                _nop_();        //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
                _nop_();
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 INT3下降沿唤醒掉电模式举例----------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/
//特别注意：在将进入掉电模式时一定要加入2-4条NOP语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条NOP语句（空语句），如本程序中所示。
//假定测试芯片的工作频率为18.432MHz

INT_CLKO        DATA    08FH                    //外部中断与时钟输出控制寄存器
INT3    BIT     P3.7                            //INT3引脚定义
//---------------------------------------
        ORG     0000H
        LJMP    MAIN                            //复位入口

        ORG     005BH                           //INT3中断入口
        LJMP    EXINT3
//---------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH
        ORL     INT_CLKO,       #20H            //(EX3 = 1)使能INT3下降沿中断
        SETB    EA

LOOP:
        MOV     PCON,   #02H                    //MCU进入掉电模式
        NOP                             //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        NOP
        SJMP    LOOP
//---------------------------------------
//中断服务程序

EXINT3:
        CPL     P1.0                            //将测试口取反
//      ANL     INT_CLKO,       #0DFH           //若需要手动清除中断标志,可先关闭中断,
                                                //此时系统会自动清除内部的中断标志
//      ORL     INT_CLKO,       #20H            //然后再开中断即可

        RETI
;---------------------------------------
        END
```

STC MCU Limited

2.3.3.7 用外部中断INT4(下降沿)唤醒掉电模式/停机模式的测试程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 INT4下降沿唤醒掉电模式举例----------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/
//特别注意：在将进入掉电模式时一定要加入2-4条_nop_()语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------
sfr     INT_CLKO    =   0x8F;       //外部中断与时钟输出控制寄存器
sbit    INT4    =   P3^0;           //INT4引脚定义
sbit    P10     =   P1^0;

//-----------------------------------------
//中断服务程序
void exint4() interrupt 16
{
    P10     =   !P10;               //将测试口取反

//  INT_CLKO    &=  0xBF;           //若需要手动清除中断标志,可先关闭中断,
                                    //此时系统会自动清除内部的中断标志
//  INT_CLKO    |=  0x40;           //然后再开中断即可
}

//-----------------------------------------------
void main()
{
    INT_CLKO |= 0x40;               //(EX4 = 1)使能INT4下降沿中断
    EA = 1;

    while (1)
    {
        PCON    =   0x02;           //MCU进入掉电模式
        _nop_();            //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        _nop_();
    }
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 INT4下降沿唤醒掉电模式举例----------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/
//特别注意：在将进入掉电模式时一定要加入2-4条NOP语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条NOP语句（空语句），如本程序中所示。
//假定测试芯片的工作频率为18.432MHz

INT_CLKO        DATA    08FH                    //外部中断与时钟输出控制寄存器
INT4            BIT     P3.0                    //INT4引脚定义

//---------------------------------------
        ORG     0000H
        LJMP    MAIN                            //复位入口

        ORG     0083H                           //INT4中断入口
        LJMP    EXINT4
//---------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH
        ORL     INT_CLKO,       #40H            //(EX4 = 1)使能INT4下降沿中断
        SETB    EA

LOOP:
        MOV     PCON,   #02H                    //MCU进入掉电模式
        NOP                             //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        NOP
        SJMP    LOOP

//---------------------------------------
//中断服务程序

EXINT4:
        CPL     P1.0                            //将测试口取反

//      ANL     INT_CLKO,       #0BFH           //若需要手动清除中断标志,可先关闭中断,
                                                //此时系统会自动清除内部的中断标志
//      ORL     INT_CLKO,       #40H            //然后再开中断即可
        RETI
;---------------------------------------
        END
```

STC MCU Limited

2.3.3.8 用CCP/PCA扩展的外部中断(下降沿+上升沿)唤醒掉电模式/停机模式的程序

1. C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 PCA扩展为外部中断唤醒掉电模式举例----------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条_nop_()语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

//本测试程序以PCA模块0为例进行说明,PCA的模块1和模块2与模块0的实用方法相同

#include "reg51.h"
#include "intrins.h"

#define    FOSC    18432000L

typedef unsigned char    BYTE;
typedef unsigned int     WORD;
typedef unsigned long    DWORD;

sfr     P_SW1    =    0xA2;                  //外设功能切换寄存器1

#define  CCP_S0  0x10                        //P_SW1.4
#define  CCP_S1  0x20                        //P_SW1.5

sfr     CCON    =    0xD8;                  //PCA控制寄存器
sbit    CCF0    =    CCON^0;                //PCA模块0中断标志
sbit    CCF1    =    CCON^1;                //PCA模块1中断标志
sbit    CR      =    CCON^6;                //PCA定时器运行控制位
sbit    CF      =    CCON^7;                //PCA定时器溢出标志
sfr     CMOD    =    0xD9;                  //PCA模式寄存器
sfr     CL      =    0xE9;                  //PCA定时器低字节
sfr     CH      =    0xF9;                  //PCA定时器高字节
sfr     CCAPM0          =       0xDA;                   //PCA模块0模式寄存器
```

STC MCU Limited

```
sfr     CCAP0L      =     0xEA;                 //PCA模块0捕获寄存器 LOW
sfr     CCAP0H      =     0xFA;                 //PCA模块0捕获寄存器 HIGH
sfr     CCAPM1      =     0xDB;                 //PCA模块1模式寄存器
sfr     CCAP1L      =     0xEB;                 //PCA模块1捕获寄存器 LOW
sfr     CCAP1H      =     0xFB;                 //PCA模块1捕获寄存器 HIGH
sfr     CCAPM2      =     0xDC;                 //PCA模块2模式寄存器
sfr     CCAP2L      =     0xEC;                 //PCA模块2捕获寄存器 LOW
sfr     CCAP2H      =     0xFC;                 //PCA模块2捕获寄存器 HIGH
sfr     PCA_PWM0    =     0xf2;                 //PCA模块0的PWM寄存器
sfr     PCA_PWM1    =     0xf3;                 //PCA模块1的PWM寄存器
sfr     PCA_PWM2    =     0xf4;                 //PCA模块2的PWM寄存器

sbit    P10     =     P1^0;

void main()
{
        ACC     =     P_SW1;
        ACC     &=    ~(CCP_S0 | CCP_S1);       //CCP_S0=0 CCP_S1=0
        P_SW1   =     ACC;                      //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      ACC     =     P_SW1;
//      ACC     &=    ~(CCP_S0 | CCP_S1);       //CCP_S0=1 CCP_S1=0
//      ACC     |=    CCP_S0;                   //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      P_SW1   =     ACC;
//
//      ACC     =     P_SW1;
//      ACC     &=    ~(CCP_S0 | CCP_S1);       //CCP_S0=0 CCP_S1=1
//      ACC     |=    CCP_S1;                   //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      P_SW1   =     ACC;

        CCON    =     0;                        //初始化PCA控制寄存器
                                                //PCA定时器停止
                                                //清除CF标志
                                                //清除模块中断标志
        CL      =     0;                        //复位PCA寄存器
        CH      =     0;
        CCAP0L  =     0;
        CCAP0H  =     0;
        CMOD    =     0x08;                     //设置PCA时钟源为系统时钟
        CCAPM0  =     0x21;                     //PCA模块0为16位捕获模式(上升沿捕获,可测从高电平开始
                                                //的整个周期),且产生捕获中断,此时CCP0上的上升沿中断
                                                //可唤醒掉电模式
//      CCAPM0  =     0x11;                     //PCA模块0为16位捕获模式(下降沿捕获,可测从低电平开始
                                                //的整个周期),且产生捕获中断,此时CCP0上的下降沿中断
                                                //可唤醒掉电模式
//      CCAPM0  =     0x31;     //PCA模块0为16位捕获模式(上升沿/下降沿捕获,可测高电平或者
```

STC MCU Limited

```
                                        //低电平宽度),且产生捕获中断,此时CCP0上的上升沿和下降沿中断
                                        //可唤醒掉电模式
        CR      =       1;              //PCA定时器开始工作
        EA      =       1;

        while (1)
        {
                PCON = 0x02;            //MCU进入掉电模式
                _nop_();                //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
                _nop_();
        }
}

void PCA_isr() interrupt 7 using 1
{
        if (CCF0)                               //判断是否为捕获中断
        {
                CCF0    =       0;
                P10     =       !P10;   //将测试口取反
        }
}
```

2. 汇编程序：

```
/*------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 PCA扩展为外部中断唤醒掉电模式举例------------------*/
/* --- 研发顾问QQ：800003751---------------- ----------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条NOP语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条NOP语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

//本测试程序以PCA模块0为例进行说明,PCA的模块1和模块2与模块0的实用方法相同

P_SW1   EQU     0A2H                    //外设功能切换寄存器1
```

STC MCU Limited

```
CCP_S0  EQU     10H                     //P_SW1.4
CCP_S1  EQU     20H                     //P_SW1.5
CCON    EQU     0D8H                    //PCA控制寄存器
CCF0    BIT     CCON.0                  //PCA模块0中断标志
CCF1    BIT     CCON.1                  //PCA模块1中断标志
CR      BIT     CCON.6                  //PCA定时器运行控制位
CF      BIT     CCON.7                  //PCA定时器溢出标志
CMOD    EQU     0D9H                    //PCA模式寄存器
CL      EQU     0E9H                    //PCA定时器低字节
CH      EQU     0F9H                    //PCA定时器高字节
CCAPM0          EQU     0DAH            //PCA模块0模式寄存器
CCAP0L          EQU     0EAH            //PCA模块0捕获寄存器 LOW
CCAP0H          EQU     0FAH            //PCA模块0捕获寄存器 HIGH
CCAPM1          EQU     0DBH            //PCA模块1模式寄存器
CCAP1L          EQU     0EBH            //PCA模块1捕获寄存器 LOW
CCAP1H          EQU     0FBH            //PCA模块1捕获寄存器 HIGH
CCAPM2          EQU     0DCH            //PCA模块2模式寄存器
CCAP2L          EQU     0ECH            //PCA模块2捕获寄存器 LOW
CCAP2H          EQU     0FCH            //PCA模块2捕获寄存器 HIGH
PCA_PWM0        EQU     0F2H            //PCA模块0的PWM寄存器
PCA_PWM1        EQU     0F3H            //PCA模块1的PWM寄存器
PCA_PWM2        EQU     0F4H            //PCA模块2的PWM寄存器

//----------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     003BH
PCA_ISR:
        PUSH    PSW
        PUSH    ACC
CKECK_CCF0:
        JNB     CCF0,   PCA_ISR_EXIT        //判断是否为捕获中断
        CLR     CCF0
        CPL     P1.0                        //将测试口取反
PCA_ISR_EXIT:
        POP     ACC
        POP     PSW
        RETI
//----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #5FH

        MOV     A,      P_SW1
        ANL     A,      #0CFH           //CCP_S0=0 CCP_S1=0
        MOV     P_SW1,  A               //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
```

STC MCU Limited

```
//      MOV     A,      P_SW1
//      ANL     A,      #0CFH           //CCP_S0=1 CCP_S1=0
//      ORL     A,      #CCP_S0         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      MOV     P_SW1,  A
//
//      MOV     A,      P_SW1
//      ANL     A,      #0CFH           //CCP_S0=0 CCP_S1=1
//      ORL     A,      #CCP_S1         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      MOV     P_SW1,  A

        MOV     CCON,   #0              //初始化PCA控制寄存器
                                        //PCA定时器停止
                                        //清除CF标志
                                        //清除模块中断标志
        CLR     A                       //
        MOV     CL,     A               //复位PCA计时器
        MOV     CH,     A               //
        MOV     CCAP0L,         A
        MOV     CCAP0H,         A
        MOV     CMOD,           #08H    //设置PCA时钟源为系统时钟
        MOV     CCAPM0,         #21H    //PCA模块0为16位捕获模式(上升沿捕获,可测从高电平开始
                                        //的整个周期),且产生捕获中断,此时CCP0上的上升沿中断
                                        //可唤醒掉电模式
//      MOV     CCAPM0,         #11H    //PCA模块0为16位捕获模式(下降沿捕获,可测从低电平开始
                                        //的整个周期),且产生捕获中断,此时CCP0上的下降沿中断
                                        //可唤醒掉电模式
//      MOV     CCAPM0,         #31H    //PCA模块0为16位捕获模式(上升沿/下降沿捕获,可测高电
                                        //平或者低电平宽度),且产生捕获中断,此时CCP0上的上升
                                        //沿和下降沿中断均可唤醒掉电模式

        SETB    CR                      //PCA定时器开始工作
        SETB    EA

LOOP:
        MOV     PCON,   #02H            //MCU进入掉电模式
        NOP                     //掉电模式被唤醒后,首先执行此语句,然后再进入中断服务程序
        NOP
        SJMP    LOOP

//---------------------------------------

        END
```

STC MCU Limited

2.3.3.9 用RxD管脚由高到低的变化唤醒掉电模式/停机模式的测试程序(C和汇编)
——现供货的STC15F2K60S2系列C版本的RxD管脚不能唤醒掉电模式/停机模式

1. C程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 RxD串行中断1唤醒掉电模式(停电模式)举例----------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条_nop_()语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------

sfr     AUXR    =   0x8e;          //辅助寄存器
sfr     T2H     =   0xd6;          //定时器2高8位
sfr     T2L     =   0xd7;          //定时器2低8位

sfr     P_SW1   =   0xA2;          //外设功能切换寄存器1

#define S1_S0   0x40               //P_SW1.6
#define S1_S1   0x80               //P_SW1.7

sbit    P10     =   P1^0;

//-----------------------------------------

void main()
{
        ACC     =   P_SW1;
        ACC     &=  ~(S1_S0 | S1_S1);       //S1_S0=0 S1_S1=0
        P_SW1   =   ACC;                    //(P3.0/RxD, P3.1/TxD)

//      ACC     =   P_SW1;
//      ACC     &=  ~(S1_S0 | S1_S1);       //S1_S0=1 S1_S1=0
//      ACC     |=  S1_S0;                   //(P3.6/RxD_2, P3.7/TxD_2)
```

```
//      P_SW1   =       ACC;
//
//      ACC     =       P_SW1;
//      ACC     &=      ~(S1_S0 | S1_S1);       //S1_S0=0 S1_S1=1
//      ACC     |=      S1_S1;                          //(P1.6/RxD_3, P1.7/TxD_3)
//      P_SW1   =       ACC;

        SCON    =       0x50;                           //8位可变波特率
        T2L     =       (65536 - (FOSC/4/BAUD));        //设置波特率重装值
        T2H     =       (65536 - (FOSC/4/BAUD))>>8;
        AUXR    =       0x14;                           //T2为1T模式, 并启动定时器2
        AUXR    |=      0x01;                           //选择定时器2为串口1的波特率发生器

        ES      =       1;
        EA      =       1;

        while (1)
        {
                PCON = 0x02;                    //MCU进入掉电模式
                _nop_();                        //掉电模式被唤醒后,直接从此语句开始向下执行,
                                                //不进入中断服务程序

                _nop_();
                P10 = !P10;                     //将测试口取反
        }
}

/*------------------------------
UART 中断服务程序
------------------------------*/
void Uart() interrupt 4 using 1
{
        if (RI)
        {
                RI = 0;                         //清除RI位
                P0 = SBUF;                      //P0显示串口数据
        }
        if (TI)
        {
                TI = 0;                         //清除TI位
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 RxD串行中断1唤醒掉电模式(停电模式)举例----------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//特别注意：在将进入掉电模式时一定要加入2-4条NOP语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条NOP语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

//-----------------------------------------

AUXR    EQU     08EH                    //辅助寄存器
T2H     DATA    0D6H                    //定时器2高8位
T2L     DATA    0D7H                    //定时器2低8位

P_SW1   EQU     0A2H                    //外设功能切换寄存器1

S1_S0   EQU     40H                     //P_SW1.6
S1_S1   EQU     80H                     //P_SW1.7

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     0023H                   //中断入口
        LJMP    UART_ISR

//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH
        MOV     A,      P_SW1
        ANL     A,      #03FH           //S1_S0=0 S1_S1=0
        MOV     P_SW1,  A               //(P3.0/RxD, P3.1/TxD)

//      MOV     A,      P_SW1
//      ANL     A,      #03FH           //S1_S0=1 S1_S1=0
//      ORL     A,      #S1_S0          //(P3.6/RxD_2, P3.7/TxD_2)
```

STC MCU Limited

```
//          MOV     P_SW1,  A
//
//          MOV     A,      P_SW1
//          ANL     A,      #03FH           //S1_S0=0 S1_S1=1
//          ORL     A,      #S1_S1          //(P1.6/RxD_3, P1.7/TxD_3)
//          MOV     P_SW1,  A

            MOV     SCON,   #50H            //8位可变波特率
            MOV     T2L,    #0D8H           //设置波特率重装值(65536-18432000/4/115200)
            MOV     T2H,    #0FFH
            MOV     AUXR,   #14H            //T2为1T模式, 并启动定时器2
            ORL     AUXR,   #01H            //选择定时器2为串口1的波特率发生器

            SETB    ES                      //打开串口中断
            SETB    A

LOOP:
            MOV     PCON,   #02H            //MCU进入掉电模式
            NOP                             //掉电模式被唤醒后,直接从此语句开始向下执行,
                                            //不进入中断服务程序
            NOP
            CPL     P1.0                    //掉电唤醒后,取反测试口
            SJMP    LOOP

;/*---------------------------
;UART 中断服务程序
;---------------------------*/
UART_ISR:
            PUSH    ACC
            PUSH    PSW
            JNB     RI,     CHECKTI         //检测RI位
            CLR     RI                      //清除RI位
            MOV     P0,     SBUF            //P0显示串口数据
CHECKTI:
            JNB     TI,     ISR_EXIT        //检测TI位
            CLR     TI                      //清除TI位
ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI
;----------------------------------------

            END
```

STC MCU Limited

2.3.3.10 用RxD2管脚由高到低的变化唤醒掉电模式/停机模式的测试程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 RxD2串行中断2唤醒掉电模式举例-----------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/
//特别注意：在将进入掉电模式时一定要加入2-4条_nop_()语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条_nop_()语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC    18432000L           //系统频率
#define   BAUD    115200              //串口波特率
#define   TM      (65536 - (FOSC/4/BAUD))

//-----------------------------------------------

sfr       AUXR    =       0x8e;       //辅助寄存器
sfr       S2CON   =       0x9a;       //UART2 控制寄存器
sfr       S2BUF   =       0x9b;       //UART2 数据寄存器
sfr       T2H     =       0xd6;       //定时器2高8位
sfr       T2L     =       0xd7;       //定时器2低8位
sfr       IE2     =       0xaf;       //中断控制寄存器2

#define   S2RI    0x01                //S2CON.0
#define   S2TI    0x02                //S2CON.1
#define   S2RB8   0x04                //S2CON.2
#define   S2TB8   0x08                //S2CON.3

sfr       P_SW2   =       0xBA;       //外设功能切换寄存器2

#define   S2_S    0x01                //P_SW2.0
```

STC MCU Limited

```
sbit    P20     =       P2^0;

//-----------------------------------------------
void main()
{
        P_SW2   &=      ~S2_S;          //S2_S=0 (P1.0/RxD2, P1.1/TxD2)
//      P_SW2   |=      S2_S;           //S2_S=1 (P4.6/RxD2_2, P4.7/TxD2_2)

        S2CON   =       0x50;           //8位可变波特率
        T2L     =       TM;             //设置波特率重装值
        T2H     =       TM>>8;
        AUXR    =       0x14;           //T2为1T模式, 并启动定时器2

        IE2     =       0x01;           //使能串口2中断
        EA      =       1;

        while (1)
        {
                PCON    =       0x02;   //MCU进入掉电模式
                _nop_();                //掉电模式被唤醒后,直接从此语句开始向下执行,
                                        //不进入中断服务程序
                _nop_();
                P20     =       !P20;   //将测试口取反
        }
}

/*----------------------------
UART2 中断服务程序
-----------------------------*/
void Uart2() interrupt 8 using 1
{
        if (S2CON & S2RI)
        {
                S2CON   &=      ~S2RI;          //清除S2RI位
                P0      =       S2BUF;          //P0显示串口数据
        }
        if (S2CON & S2TI)
        {
                S2CON   &=      ~S2TI;          //清除S2TI位
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 RxD2串行中断2唤醒掉电模式举例-------------------------*/
/* --- 研发顾问QQ：800003751---------------- ------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/
//特别注意：在将进入掉电模式时一定要加入2-4条NOP语句（空语句），即一定要在设置MCU进入
//掉电模式的语句后加2-4条NOP语句（空语句），如本程序中所示。

//假定测试芯片的工作频率为18.432MHz

AUXR    EQU     08EH                    //辅助寄存器
S2CON   EQU     09AH                    //UART2 控制寄存器
S2BUF   EQU     09BH                    //UART2 数据寄存器
T2H     DATA    0D6H                    //定时器2高8位
T2L     DATA    0D7H                    //定时器2低8位
IE2     EQU     0AFH                    //中断控制寄存器2

P_SW2   EQU     0BAH                    //外设功能切换寄存器2

S2_S    EQU     01H                     //P_SW2.0

S2RI    EQU     01H                     //S2CON.0
S2TI    EQU     02H                     //S2CON.1
S2RB8   EQU     04H                     //S2CON.2
S2TB8   EQU     08H                     //S2CON.3

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                            //复位入口

        ORG     0043H                           //中断入口
        LJMP    UART2_ISR
//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ANL     P_SW2,  #NOT S2_S               //S2_S=0 (P1.0/RxD2, P1.1/TxD2)
//      ORL     P_SW2,  #S2_S                   //S2_S=1 (P4.6/RxD2_2, P4.7/TxD2_2)
```

STC MCU Limited

```
        MOV     S2CON,  #50H            //8位可变波特率
        MOV     T2L,    #0D8H           //设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H            //T2为1T模式, 并启动定时器2

        ORL     IE2,    #01H            //使能串口2中断
        SETB    EA

LOOP:
        MOV     PCON,   #02H            //MCU进入掉电模式
        NOP                             //掉电模式被唤醒后,直接从此语句开始向下执行,
                                        //不进入中断服务程序
        NOP
        CPL     P1.0                    //掉电唤醒后,取反测试口
        SJMP    LOOP

;/*---------------------------
;UART2 中断服务程序
;---------------------------*/
UART2_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     A,      S2CON           ;读取UART2控制寄存器
        JNB     ACC.0,  CHECKTI         ;检测S2RI位
        ANL     S2CON,  #NOT S2RI       ;清除S2RI位
        MOV     P0,     S2BUF           ;P0显示串口数据
CHECKTI:                                ;
        MOV     A,      S2CON           ;读取UART2控制寄存器
        JNB     ACC.1,  ISR_EXIT        ;检测S2TI位
        ANL     S2CON,  #NOT S2TI       ;清除S2TI位
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

;----------------------------------------

        END
```

STC MCU Limited

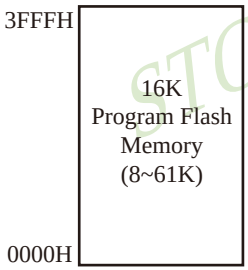
第3章 存储器和特殊功能寄存器(SFRs)

STC15系列单片机的程序存储器和数据存储器是各自独立编址的. STC15系列单片机的所有程序存储器都是片上Flash存储器，不能访问外部程序存储器，因为没有访问外部程序存储器的总线。

STC15系列单片机内部集成了大容量的数据存储器，如STC15W4K60S4系列单片机内部有4096字节的数据存储器、STC15F2K60S2系列单片机内部有2048字节的数据存储器等。STC-15W4K60S4系列单片机内部的4096字节数据存储器在物理和逻辑上都分为两个地址空间:内部RAM(256字节)和内部扩展RAM(3840字节)。其中内部RAM的高128字节的数据存储器与特殊功能寄存器(SFRs)貌似地址重叠，实际使用时通过不同的寻址方式加以区分。另外，STC15系列40-pin及其以上的单片机还可以访问在片外扩展的64KB外部数据存储器。

3.1 程序存储器

程序存储器用于存放用户程序、数据和表格等信息。以STC15W4K60S4系列单片机为例，STC15W4K60S4系列单片内部集成了8K~61K字节的Flash程序存储器。STC15W4K60S4系列各种型号单片机的程序Flash存储器的地址如下表所示。



STC15F4K16S4单片机程序存储器

Type	Program Memory
STC15W4K08S4	0000H~1FFFH (8K)
STC15W4K16S4	0000H~3FFFH (16K)
STC15W4K24S4	0000H~5FFFH (24K)
STC15W4K32S4	0000H~7FFFH (32K)
STC15W4K40S4	0000H~9FFFH (40K)
STC15W4K48S4	0000H~0BFFFH (48K)
STC15W4K56S4	0000H~0DFFFH (56K)
STC15W4K60S4	0000H~0EFFFH (60K)
IAP15W4K61S4	0000H~0F3FFH (61K)

单片机复位后，程序计数器(PC)的内容为0000H，从0000H单元开始执行程序。另外中断服务程序的入口地址(又称中断向量)也位于程序存储器单元。在程序存储器中，每个中断都有一个固定的入口地址，当中断发生并得到响应后，单片机就会自动跳转到相应的中断入口地址去执行程序。外部中断0的中断服务程序的入口地址是0003H，定时器/计数器0中断服务程序的入口地址是000BH,外部中断1的中断服务程序的入口地址是0013H，定时器/计数器1的中断服务程序的入口地址是001BH等。更多的中断服务程序的入口地址(中断向量)见单独的中断章节。由于相邻中断入口地址的间隔区间(8个字节)有限，一般情况下无法保存完整的中断服务程序，因此，一般在中断响应的地址区域存放一条无条件转移指令，指向真正存放中断服务程序的空间去执行。

程序Flash存储器可在线反复编程擦写10万次以上，提高了使用的灵活性和方便性。

STC MCU Limited

3.2 数据存储器(SRAM)

下表总结了STC15系列单片机内部数据存储器(SRAM)的空间大小以及是否可以扩展片外数据存储器:

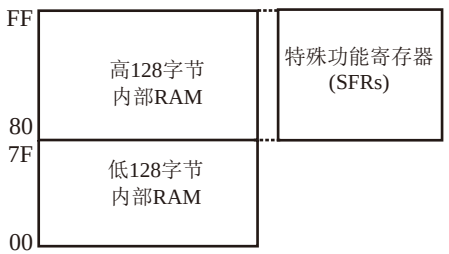
数据存储器 / 单片机型号	内部集成的总SRAM大小（字节/Byte）	内部扩展RAM空间大小（字节/Byte）	是否可以片外扩展64KB的外部数据存储器
STC15W4K60S4系列	4K (256 <idata> + 3840 <xdata>)	3840	可以
STC15F2K60S2系列	2K (256 <idata> + 1792 <xdata>)	1792	可以
STC15W1K16S系列	1K (256 <idata> +768 <xdata>)	768	可以
STC15W408S系列	512 (256 <idata> +256 <xdata>)	256	可以
STC15W408AS系列	512 (256 <idata> +256 <xdata>)	256	不可以
STC15W201S系列	256 <idata>	无内部扩展RAM	不可以
STC15F408AD系列	512 (256 <idata> +256 <xdata>)	256	不可以
STC15F101W系列	128 <idata>	无内部扩展RAM	不可以

STC15系列单片机内部集成的RAM可用于存放程序执行的中间结果和过程数据。以STC15W4K60S4系列单片机为例，STC15W4K60S4系列单片机内部集成了4096字节RAM内部数据存储器，其在物理和逻辑上都分为两个地址空间:内部RAM(256字节)和内部扩展RAM(3840字节)。此外，STC15系列40-pin及其以上的单片机还可以访问在片外扩展的64KB外部数据存储器。

3.2.1 内部RAM

内部RAM共256字节，可分为3**个部分：低128字节RAM(与传统8051兼容)、高128字节RAM(Intel在8052中扩展了高128字节RAM)**及特殊功能寄存器区。低128字节的数据存储器既可直接寻址也可间接寻址。高128字节RAM与特殊功能寄存器区貌似共用相同的地址范围，都使用80H~FFH,地址空间虽然貌似重叠，但物理上是独立的，使用时通过不同的寻址方式加以区分。高128字节RAM只能间接寻址，特殊功能寄存器区只可直接寻址。

内部RAM的结构如下图所示，地址范围是00H~FFH。



内部 RAM

低128字节的内部RAM

低128字节RAM也称通用RAM区。通用RAM区又可分为工作寄存器组区，可位寻址区，用户RAM区和堆栈区。工作寄存器组区地址从00H~1FH共32B(字节)单元，分为4组(每一组称为一个寄存器组)，每组包含8个8位的工作寄存器，编号均为R0 ~ R7，但属于不同的物理空间。通过使用工作寄存器组，可以提高运算速度。R0~R7是常用的寄存器，提供4组是因为1组往往不够用。程序状态字PSW寄存器中的RS1和RS0组合决定当前使用的工作寄存器组。见下面PSW寄存器的介绍。可位寻址区的地址从20H ~ 2FH共16个字节单元。20H~2FH单元既可向普通RAM单元一样按字节存取，也可以对单元中的任何一位单独存取，共128位，所对应的地址范围是00H~7FH。位地址范围是00H~7FH，内部RAM低128字节的地址也是00H~7FH；从外表看，二者地址是一样的，实际上二者具有本质的区别；位地址指向的是一个位，而字节地址指向的是一个字节单元，在程序中使用不同的指令区分。内部RAM中的30H~FFH单元是用户RAM和堆栈区。一个8位的堆栈指针(SP)，用于指向堆栈区。单片机复位后，堆栈指针SP为07H，指向了工作寄存器组0中的R7，因此，用户初始化程序都应对SP设置初值，一般设置在80H以后的单元为宜。

STC MCU Limited

PSW ： 程序状态字寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	name	CY	AC	F0	RS1	RS0	OV	-	P

CY ： 标志位。进行加法运算时，当最高位即B7位有进位，或执行减法运算最高位有借位时，CY为1；反之为0

AC ： 进位辅助位。进行加法运算时，当B3位有进位，或执行减法运算B3有借位时，AC为1；反之为0。设置辅助进位标志AC的目的是为了便于BCD码加法、减法运算的调整。

F0 ： 用户标志位。

RS1、RS0: 工作寄存器组的选择位。如下表

RS1	RS0	当前使用的工作寄存器组(R0~R7)
0	0	0组(00H~07H)
0	1	1组(08H~0FH)
1	0	2组(10H~17H)
1	1	3组(18H~1FH)

OV : 溢出标志位.

B1 ： 保留位

P ： 奇偶标志位。该标志位始终体现累加器ACC中1的个数的奇偶性。如果累加器ACC中1的个数为奇数，则P置1；当累加器ACC中的个数为偶数(包括0个)时，P位为0

堆栈指针(SP):

堆栈指针是一个8位专用寄存器。它指示出堆栈顶部在内部RAM块中的位置。系统复位后，SP初始化位07H，使得堆栈事实上由08H单元开始，考虑08H~1FH单元分别属于工作寄存器组1~3，若在程序设计中用到这些区，则最好把SP值改变为80H或更大的值为宜。STC15系列单片机的堆栈是向上生长的，即将数据压入堆栈后，SP内容增大。

3.2.2 内部扩展RAM / XRAM / AUX-RAM及测试程序

STC15W4K60S4系列单片机片内除了集成256字节的内部RAM外，还集成了3840字节的扩展RAM，地址范围是0000H~0EFFH. 访问内部扩展RAM的方法和传统8051单片机访问外部扩展RAM的方法相同，但是不影响P0口(数据总线和高八位地址总线)、P2口(低八位地址总线)、$\overline{WR}$/P4.2、$\overline{RD}$/P4.4和ALE/P4.5。在汇编语言中，内部扩展RAM通过MOVX指令访问，即使用"MOVX　@DPTR"或者"MOVX　@Ri"指令访问。在C语言中，可使用xdata声明存储类型即可，如"unsigned　char　xdata　i=0;"。

单片机内部扩展RAM是否可以访问受辅助寄存器AUXR(地址为8EH)中的EXTRAM位控制。

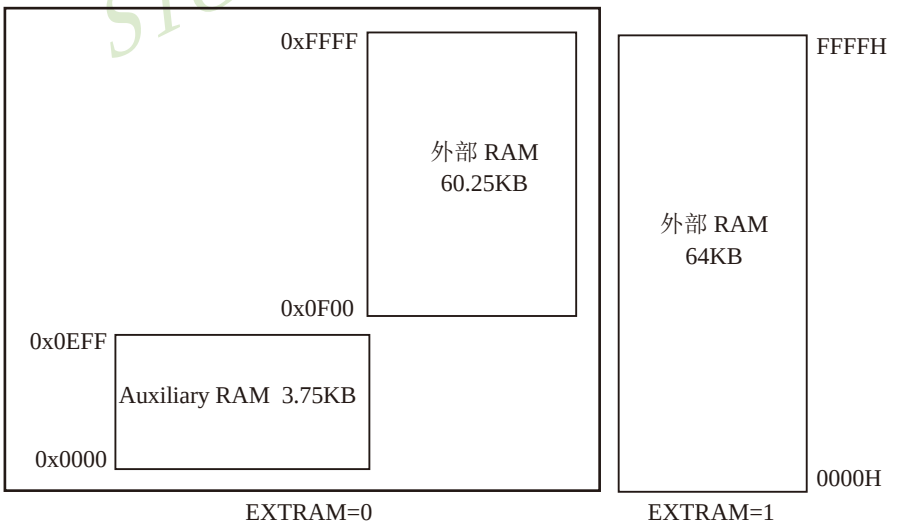
STC15W4K60S4系列单片机8051单片机 扩展RAM管理及禁止ALE输出 特殊功能寄存器

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR	8EH	Auxiliary Register	T0x12	T1x12	UAR_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2	0000,0001

EXTRAM: Internal/External RAM access 内部/外部RAM存取

0: 内部扩展的EXT_RAM可以存取.
STC15W4K60S4系列单片机
在00H到EFFH单元(3840字节),使用MOVX　@DPTR指令访问,超过F00H(含F00H单元)的地址空间总是访问外部数据存储器，MOVX　@Ri只能访问00H到FFH单元

1: External data memory access. 外部数据存储器存取
禁止访问内部扩展RAM，此时MOVX　@DPTR/MOVX　@Ri的使用同普通8052单片机



STC MCU Limited

应用示例供参考（汇编）：

```
访问内部扩展的EXTRAM
;新增特殊功能寄存器声明(汇编方式)
AUXR    DATA    8EH     ;或者用        AUXR    EQU     8EH     定义
MOV     AUXR,   #00000000B      ;EXTRAM位清为“0”，其实上电复位时此位就为“0”.
;MOVX   A,      @DPTR   / MOVX @DPTR, A 指令可访问内部扩展的EXTRAM
               ;STC15W4K60S4系列为(00H - EFFH,共3840字节)

;MOVX   A,      @Ri     / MOVX A,       @Ri 指令可直接访问内部扩展的EXTRAM
               ;使用此指令 只能访问内部扩展的EXTRAM(00H - FFH,共256字节)
;写芯片内部扩展的EXTRAM
        MOV     DPTR,   #address
        MOV     A,      #value
        MOVX    @DPTR,  A
;读芯片内部扩展的EXTRAM
        MOV     DPTR,   #address
        MOVX    A,      @DPTR
STC15W4K60S4系列
; 如果 #address < F00H，则在EXTRAM位为”0”时，访问物理上在内部，逻辑上在外部的
        此EXTRAM
; 如果 #address >= F00H，则总是访问物理上外部扩展的RAM或I/O空间（F00H--FFFFH）
```

STC MCU Limited

禁止访问内部扩展的EXTRAM，以防冲突

MOV　　AUXR,　#00000010B;　EXTRAM控制位设置为”1”,禁止访问EXTRAM,以防冲突
有些用户系统因为外部扩展了I/O 或者用片选去选多个RAM 区,有时与此内部扩展的EXTRAM逻辑地址上有冲突，将此位设置为”1”,禁止访问此内部扩展的EXTRAM就可以了.

大实话：

其实不用设置AUXR寄存器即可直接用MOVX @DPTR指令访问此内部扩展的EXTRAM,超过此RAM空间,将访问片外单元.如果系统外扩了SRAM,而实际使用的空间小于3840字节,则可直接将此SRAM省去,比如省去STC62WV256,IS62C256,UT6264等.

应用示例供参考（C语言）：

```
/*      访问内部扩展的EXTRAM */
/*  STC15W4K60S4系列单片机为(00H - EFFH, 共3840字节扩展的EXTRAM) */
/* 新增特殊功能寄存器声明(C 语言方式)  */
sfr  AUXR = 0x8e          /*如果不需设置AUXR就不用声明AUXR     */
AUXR = 0x00;              /*0000,0000  EXTRAM位清0, 其实上电复位时此位就为0 */
unsigned char  xdata  sum,      loop_counter, test_array[128];
/*  将变量声明成 xdata 即可直接访问此内部扩展的EXTRAM*/
```

```
/*  写芯片内部扩展的EXTRAM   */
    sum     =       0;
    loop_counter  =       128;
    test_array[0] =       5;

/*  读芯片内部扩展的EXTRAM          */
    sum     =       test_array[0];
```

如果 #address < F00H，则在EXTRAM位为”0”时，访问物理上在内部，逻辑上在外部的此EXTRAM

如果#address>=F00H，则总是访问物理上外部扩展的RAM或I/O空间（F00H-FFFFH）

禁止访问内部扩展的EXTRAM，以防冲突

```
AUXR  = 0x02;     /* 0000,0010,EXTRAM位设为”1”,禁止访问EXTRAM,以防冲突*/
```

有些用户系统因为外部扩展了I/O 或者用片选去选多个RAM 区,有时与此内部扩展的EXTRAM逻辑上有冲突，将此位设置为”1”， 禁止访问此内部扩展的EXTRAM就可以了.

STC MCU Limited

3.2.3 使用内部扩展RAM的测试程序

STC15系列单片机内部扩展RAM演示程序,本程序应用2048的内部扩展RAM

```
;/* ---  STC International Limited ------------------------------------------------------ */
;/* ---  STC  设计 2006/1/6   V1.0 ------------------------------------------------------ */
;/* ---  演示 STC15系列单片机 MCU 内部扩展RAM演示程序----------------------*/
;/* ---  研发顾问QQ：800003751----------------------------------------------------------*/
;/* ---  Fax: 0755-82905966 ------------------------------------------------------------- */
;/* ---  Tel: 0755-82948409 ------------------------------------------------------------- */
;/* ---  Web: www.STCMCU.com ------------------------------------------------------------ */
;/* ---  本演示程序在STC 15系列 ISP的下载编程工具上测试通过 ------------------- */
;/* ---  如果要在程序中使用该程序,请在程序中注明使用了STC的资料及程序 --- */
;/* ---  如果要在文章中引用该程序,请在文章中注明使用了STC的资料及程序 --- */

#include  <reg52.h>
#include  <intrins.h>                                   /* use _nop_() function */

sfr       AUXR            =       0x8e;

sbit      ERROR_LED       =       P1^5;
sbit      OK_LED          =       P1^7;

void main()
{
    unsigned int array_point        =       0;

  /* 测试数组 Test_array_one[1024],  Test_array_two[1024]*/
    unsigned char xdata Test_array_one[1024]        =
    {
        0x00,   0x01,   0x02,   0x03,   0x04,   0x05,   0x06,   0x07,
        0x08,   0x09,   0x0a,   0x0b,   0x0c,   0x0d,   0x0e,   0x0f,
        0x10,   0x11,   0x12,   0x13,   0x14,   0x15,   0x16,   0x17,
        0x18,   0x19,   0x1a,   0x1b,   0x1c,   0x1d,   0x1e,   0x1f,
        0x20,   0x21,   0x22,   0x23,   0x24,   0x25,   0x26,   0x27,
        0x28,   0x29,   0x2a,   0x2b,   0x2c,   0x2d,   0x2e,   0x2f,
        0x30,   0x31,   0x32,   0x33,   0x34,   0x35,   0x36,   0x37,
        0x38,   0x39,   0x3a,   0x3b,   0x3c,   0x3d,   0x3e,   0x3f,
        0x40,   0x41,   0x42,   0x43,   0x44,   0x45,   0x46,   0x47,
        0x48,   0x49,   0x4a,   0x4b,   0x4c,   0x4d,   0x4e,   0x4f,
        0x50,   0x51,   0x52,   0x53,   0x54,   0x55,   0x56,   0x57,
        0x58,   0x59,   0x5a,   0x5b,   0x5c,   0x5d,   0x5e,   0x5f,
        0x60,   0x61,   0x62,   0x63,   0x64,   0x65,   0x66,   0x67,
        0x68,   0x69,   0x6a,   0x6b,   0x6c,   0x6d,   0x6e,   0x6f,
```

STC MCU Limited

```
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f,
0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf,
0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf,
0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7,
0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
0xff, 0xfe, 0xfd, 0xfc, 0xfb, 0xfa, 0xf9, 0xf8,
0xf7, 0xf6, 0xf5, 0xf4, 0xf3, 0xf2, 0xf1, 0xf0,
0xef, 0xee, 0xed, 0xec, 0xeb, 0xea, 0xe9, 0xe8,
0xe7, 0xe6, 0xe5, 0xe4, 0xe3, 0xe2, 0xe1, 0xe0,
0xdf, 0xde, 0xdd, 0xdc, 0xdb, 0xda, 0xd9, 0xd8,
0xd7, 0xd6, 0xd5, 0xd4, 0xd3, 0xd2, 0xd1, 0xd0,
0xcf, 0xce, 0xcd, 0xcc, 0xcb, 0xca, 0xc9, 0xc8,
0xc7, 0xc6, 0xc5, 0xc4, 0xc3, 0xc2, 0xc1, 0xc0,
0xbf, 0xbe, 0xbd, 0xbc, 0xbb, 0xba, 0xb9, 0xb8,
0xb7, 0xb6, 0xb5, 0xb4, 0xb3, 0xb2, 0xb1, 0xb0,
0xaf, 0xae, 0xad, 0xac, 0xab, 0xaa, 0xa9, 0xa8,
0xa7, 0xa6, 0xa5, 0xa4, 0xa3, 0xa2, 0xa1, 0xa0,
0x9f, 0x9e, 0x9d, 0x9c, 0x9b, 0x9a, 0x99, 0x98,
0x97, 0x96, 0x95, 0x94, 0x93, 0x92, 0x91, 0x90,
0x8f, 0x8e, 0x8d, 0x8c, 0x8b, 0x8a, 0x89, 0x88,
0x87, 0x86, 0x85, 0x84, 0x83, 0x82, 0x81, 0x80,
0x7f, 0x7e, 0x7d, 0x7c, 0x7b, 0x7a, 0x79, 0x78,
0x77, 0x76, 0x75, 0x74, 0x73, 0x72, 0x71, 0x70,
0x6f, 0x6e, 0x6d, 0x6c, 0x6b, 0x6a, 0x69, 0x68,
0x67, 0x66, 0x65, 0x64, 0x63, 0x62, 0x61, 0x60,
0x5f, 0x5e, 0x5d, 0x5c, 0x5b, 0x5a, 0x59, 0x58,
0x57, 0x56, 0x55, 0x54, 0x53, 0x52, 0x51, 0x50,
0x4f, 0x4e, 0x4d, 0x4c, 0x4b, 0x4a, 0x49, 0x48,
0x47, 0x46, 0x45, 0x44, 0x43, 0x42, 0x41, 0x40,
0x3f, 0x3e, 0x3d, 0x3c, 0x3b, 0x3a, 0x39, 0x38,
0x37, 0x36, 0x35, 0x34, 0x33, 0x32, 0x31, 0x30,
```

STC MCU Limited

```
0x2f, 0x2e, 0x2d, 0x2c, 0x2b, 0x2a, 0x29, 0x28,
0x27, 0x26, 0x25, 0x24, 0x23, 0x22, 0x21, 0x20,
0x1f, 0x1e, 0x1d, 0x1c, 0x1b, 0x1a, 0x19, 0x18,
0x17, 0x16, 0x15, 0x14, 0x13, 0x12, 0x11, 0x10,
0x0f, 0x0e, 0x0d, 0x0c, 0x0b, 0x0a, 0x09, 0x08,
0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00,
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f,
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47,
0x48, 0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f,
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57,
0x58, 0x59, 0x5a, 0x5b, 0x5c, 0x5d, 0x5e, 0x5f,
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f,
0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf,
0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf,
0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7,
0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
0xff, 0xfe, 0xfd, 0xfc, 0xfb, 0xfa, 0xf9, 0xf8,
0xf7, 0xf6, 0xf5, 0xf4, 0xf3, 0xf2, 0xf1, 0xf0,
0xef, 0xee, 0xed, 0xec, 0xeb, 0xea, 0xe9, 0xe8,
0xe7, 0xe6, 0xe5, 0xe4, 0xe3, 0xe2, 0xe1, 0xe0,
0xdf, 0xde, 0xdd, 0xdc, 0xdb, 0xda, 0xd9, 0xd8,
0xd7, 0xd6, 0xd5, 0xd4, 0xd3, 0xd2, 0xd1, 0xd0,
```

STC MCU Limited

```
    0xcf,   0xce,   0xcd,   0xcc,   0xcb,   0xca,   0xc9,   0xc8,
    0xc7,   0xc6,   0xc5,   0xc4,   0xc3,   0xc2,   0xc1,   0xc0,
    0xbf,   0xbe,   0xbd,   0xbc,   0xbb,   0xba,   0xb9,   0xb8,
    0xb7,   0xb6,   0xb5,   0xb4,   0xb3,   0xb2,   0xb1,   0xb0,
    0xaf,   0xae,   0xad,   0xac,   0xab,   0xaa,   0xa9,   0xa8,
    0xa7,   0xa6,   0xa5,   0xa4,   0xa3,   0xa2,   0xa1,   0xa0,
    0x9f,   0x9e,   0x9d,   0x9c,   0x9b,   0x9a,   0x99,   0x98,
    0x97,   0x96,   0x95,   0x94,   0x93,   0x92,   0x91,   0x90,
    0x8f,   0x8e,   0x8d,   0x8c,   0x8b,   0x8a,   0x89,   0x88,
    0x87,   0x86,   0x85,   0x84,   0x83,   0x82,   0x81,   0x80,
    0x7f,   0x7e,   0x7d,   0x7c,   0x7b,   0x7a,   0x79,   0x78,
    0x77,   0x76,   0x75,   0x74,   0x73,   0x72,   0x71,   0x70,
    0x6f,   0x6e,   0x6d,   0x6c,   0x6b,   0x6a,   0x69,   0x68,
    0x67,   0x66,   0x65,   0x64,   0x63,   0x62,   0x61,   0x60,
    0x5f,   0x5e,   0x5d,   0x5c,   0x5b,   0x5a,   0x59,   0x58,
    0x57,   0x56,   0x55,   0x54,   0x53,   0x52,   0x51,   0x50,
    0x4f,   0x4e,   0x4d,   0x4c,   0x4b,   0x4a,   0x49,   0x48,
    0x47,   0x46,   0x45,   0x44,   0x43,   0x42,   0x41,   0x40,
    0x3f,   0x3e,   0x3d,   0x3c,   0x3b,   0x3a,   0x39,   0x38,
    0x37,   0x36,   0x35,   0x34,   0x33,   0x32,   0x31,   0x30,
    0x2f,   0x2e,   0x2d,   0x2c,   0x2b,   0x2a,   0x29,   0x28,
    0x27,   0x26,   0x25,   0x24,   0x23,   0x22,   0x21,   0x20,
    0x1f,   0x1e,   0x1d,   0x1c,   0x1b,   0x1a,   0x19,   0x18,
    0x17,   0x16,   0x15,   0x14,   0x13,   0x12,   0x11,   0x10,
    0x0f,   0x0e,   0x0d,   0x0c,   0x0b,   0x0a,   0x09,   0x08,
    0x07,   0x06,   0x05,   0x04,   0x03,   0x02,   0x01,   0x00
};

unsigned char xdata Test_array_two[1024]      =
{
    0x00,   0x01,   0x02,   0x03,   0x04,   0x05,   0x06,   0x07,
    0x08,   0x09,   0x0a,   0x0b,   0x0c,   0x0d,   0x0e,   0x0f,
    0x10,   0x11,   0x12,   0x13,   0x14,   0x15,   0x16,   0x17,
    0x18,   0x19,   0x1a,   0x1b,   0x1c,   0x1d,   0x1e,   0x1f,
    0x20,   0x21,   0x22,   0x23,   0x24,   0x25,   0x26,   0x27,
    0x28,   0x29,   0x2a,   0x2b,   0x2c,   0x2d,   0x2e,   0x2f,
    0x30,   0x31,   0x32,   0x33,   0x34,   0x35,   0x36,   0x37,
    0x38,   0x39,   0x3a,   0x3b,   0x3c,   0x3d,   0x3e,   0x3f,
    0x40,   0x41,   0x42,   0x43,   0x44,   0x45,   0x46,   0x47,
    0x48,   0x49,   0x4a,   0x4b,   0x4c,   0x4d,   0x4e,   0x4f,
    0x50,   0x51,   0x52,   0x53,   0x54,   0x55,   0x56,   0x57,
    0x58,   0x59,   0x5a,   0x5b,   0x5c,   0x5d,   0x5e,   0x5f,
    0x60,   0x61,   0x62,   0x63,   0x64,   0x65,   0x66,   0x67,
    0x68,   0x69,   0x6a,   0x6b,   0x6c,   0x6d,   0x6e,   0x6f,
    0x70,   0x71,   0x72,   0x73,   0x74,   0x75,   0x76,   0x77,
```

STC MCU Limited

```
0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f,
0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf,
0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf,
0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7,
0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
0xff, 0xfe, 0xfd, 0xfc, 0xfb, 0xfa, 0xf9, 0xf8,
0xf7, 0xf6, 0xf5, 0xf4, 0xf3, 0xf2, 0xf1, 0xf0,
0xef, 0xee, 0xed, 0xec, 0xeb, 0xea, 0xe9, 0xe8,
0xe7, 0xe6, 0xe5, 0xe4, 0xe3, 0xe2, 0xe1, 0xe0,
0xdf, 0xde, 0xdd, 0xdc, 0xdb, 0xda, 0xd9, 0xd8,
0xd7, 0xd6, 0xd5, 0xd4, 0xd3, 0xd2, 0xd1, 0xd0,
0xcf, 0xce, 0xcd, 0xcc, 0xcb, 0xca, 0xc9, 0xc8,
0xc7, 0xc6, 0xc5, 0xc4, 0xc3, 0xc2, 0xc1, 0xc0,
0xbf, 0xbe, 0xbd, 0xbc, 0xbb, 0xba, 0xb9, 0xb8,
0xb7, 0xb6, 0xb5, 0xb4, 0xb3, 0xb2, 0xb1, 0xb0,
0xaf, 0xae, 0xad, 0xac, 0xab, 0xaa, 0xa9, 0xa8,
0xa7, 0xa6, 0xa5, 0xa4, 0xa3, 0xa2, 0xa1, 0xa0,
0x9f, 0x9e, 0x9d, 0x9c, 0x9b, 0x9a, 0x99, 0x98,
0x97, 0x96, 0x95, 0x94, 0x93, 0x92, 0x91, 0x90,
0x8f, 0x8e, 0x8d, 0x8c, 0x8b, 0x8a, 0x89, 0x88,
0x87, 0x86, 0x85, 0x84, 0x83, 0x82, 0x81, 0x80,
0x7f, 0x7e, 0x7d, 0x7c, 0x7b, 0x7a, 0x79, 0x78,
0x77, 0x76, 0x75, 0x74, 0x73, 0x72, 0x71, 0x70,
0x6f, 0x6e, 0x6d, 0x6c, 0x6b, 0x6a, 0x69, 0x68,
0x67, 0x66, 0x65, 0x64, 0x63, 0x62, 0x61, 0x60,
0x5f, 0x5e, 0x5d, 0x5c, 0x5b, 0x5a, 0x59, 0x58,
0x57, 0x56, 0x55, 0x54, 0x53, 0x52, 0x51, 0x50,
0x4f, 0x4e, 0x4d, 0x4c, 0x4b, 0x4a, 0x49, 0x48,
0x47, 0x46, 0x45, 0x44, 0x43, 0x42, 0x41, 0x40,
0x3f, 0x3e, 0x3d, 0x3c, 0x3b, 0x3a, 0x39, 0x38,
0x37, 0x36, 0x35, 0x34, 0x33, 0x32, 0x31, 0x30,
0x2f, 0x2e, 0x2d, 0x2c, 0x2b, 0x2a, 0x29, 0x28,
```

STC MCU Limited

```
0x27, 0x26, 0x25, 0x24, 0x23, 0x22, 0x21, 0x20,
0x1f, 0x1e, 0x1d, 0x1c, 0x1b, 0x1a, 0x19, 0x18,
0x17, 0x16, 0x15, 0x14, 0x13, 0x12, 0x11, 0x10,
0x0f, 0x0e, 0x0d, 0x0c, 0x0b, 0x0a, 0x09, 0x08,
0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00,
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f,
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47,
0x48, 0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f,
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57,
0x58, 0x59, 0x5a, 0x5b, 0x5c, 0x5d, 0x5e, 0x5f,
0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f,
0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf,
0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf,
0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7,
0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
0xff, 0xfe, 0xfd, 0xfc, 0xfb, 0xfa, 0xf9, 0xf8,
0xf7, 0xf6, 0xf5, 0xf4, 0xf3, 0xf2, 0xf1, 0xf0,
0xef, 0xee, 0xed, 0xec, 0xeb, 0xea, 0xe9, 0xe8,
0xe7, 0xe6, 0xe5, 0xe4, 0xe3, 0xe2, 0xe1, 0xe0,
0xdf, 0xde, 0xdd, 0xdc, 0xdb, 0xda, 0xd9, 0xd8,
0xd7, 0xd6, 0xd5, 0xd4, 0xd3, 0xd2, 0xd1, 0xd0,
0xcf, 0xce, 0xcd, 0xcc, 0xcb, 0xca, 0xc9, 0xc8,
```

STC MCU Limited

```
        0xc7,   0xc6,   0xc5,   0xc4,   0xc3,   0xc2,   0xc1,   0xc0,
        0xbf,   0xbe,   0xbd,   0xbc,   0xbb,   0xba,   0xb9,   0xb8,
        0xb7,   0xb6,   0xb5,   0xb4,   0xb3,   0xb2,   0xb1,   0xb0,
        0xaf,   0xae,   0xad,   0xac,   0xab,   0xaa,   0xa9,   0xa8,
        0xa7,   0xa6,   0xa5,   0xa4,   0xa3,   0xa2,   0xa1,   0xa0,
        0x9f,   0x9e,   0x9d,   0x9c,   0x9b,   0x9a,   0x99,   0x98,
        0x97,   0x96,   0x95,   0x94,   0x93,   0x92,   0x91,   0x90,
        0x8f,   0x8e,   0x8d,   0x8c,   0x8b,   0x8a,   0x89,   0x88,
        0x87,   0x86,   0x85,   0x84,   0x83,   0x82,   0x81,   0x80,
        0x7f,   0x7e,   0x7d,   0x7c,   0x7b,   0x7a,   0x79,   0x78,
        0x77,   0x76,   0x75,   0x74,   0x73,   0x72,   0x71,   0x70,
        0x6f,   0x6e,   0x6d,   0x6c,   0x6b,   0x6a,   0x69,   0x68,
        0x67,   0x66,   0x65,   0x64,   0x63,   0x62,   0x61,   0x60,
        0x5f,   0x5e,   0x5d,   0x5c,   0x5b,   0x5a,   0x59,   0x58,
        0x57,   0x56,   0x55,   0x54,   0x53,   0x52,   0x51,   0x50,
        0x4f,   0x4e,   0x4d,   0x4c,   0x4b,   0x4a,   0x49,   0x48,
        0x47,   0x46,   0x45,   0x44,   0x43,   0x42,   0x41,   0x40,
        0x3f,   0x3e,   0x3d,   0x3c,   0x3b,   0x3a,   0x39,   0x38,
        0x37,   0x36,   0x35,   0x34,   0x33,   0x32,   0x31,   0x30,
        0x2f,   0x2e,   0x2d,   0x2c,   0x2b,   0x2a,   0x29,   0x28,
        0x27,   0x26,   0x25,   0x24,   0x23,   0x22,   0x21,   0x20,
        0x1f,   0x1e,   0x1d,   0x1c,   0x1b,   0x1a,   0x19,   0x18,
        0x17,   0x16,   0x15,   0x14,   0x13,   0x12,   0x11,   0x10,
        0x0f,   0x0e,   0x0d,   0x0c,   0x0b,   0x0a,   0x09,   0x08,
        0x07,   0x06,   0x05,   0x04,   0x03,   0x02,   0x01,   0x00
    };
    ERROR_LED = 1;
    OK_LED = 1;
    for(array_point=0; array_point<1024; array_point++)
    {
        if(Test_array_one[array_point]!=Test_array_two[array_point])
        {
                ERROR_LED   =   0;
                OK_LED      =   1;
                break;
        }
        else
        {
                OK_LED      =   0;
                ERROR_LED   =   1;
        }
    }
    while(1);
}
```

STC MCU Limited

3.2.3 外部64K数据总线 — 可外部扩展64K字节的数据存储器或外围设备

STC15系列40-pin及其以上的单片机具有扩展64KB外部数据存储器和I/O口的能力。访问外部数据存储器期间，$\overline{WR}$或$\overline{RD}$信号要有效。STC15系列单片机新增了一个控制外部64K字节数据总线速度的特殊功能寄存器—BUS_SPEED，该寄存器的格式如下。

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
BUS_SPEED	A1H	Bus-Speed Control	-	-	-	-	-	-	EXRTS[1:0]		xxxx,xx10

EXRTS (**E**xtend **R**am **T**iming **S**elector)

0 0： Setup / Hold / Read and Write Duty ← 1 clock cycle; EXRAC ← 1
0 1： Setup / Hold / Read and Write Duty ← 2 clock cycle; EXRAC ← 2
1 0： Setup / Hold / Read and Write Duty ← 4 clock cycle; EXRAC ← 4
1 1： Setup / Hold / Read and Write Duty ← 8 clock cycle; EXRAC ← 8

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR	8EH	Auxiliary Register	T0x12	T1x12	UAR_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2	0000,0001

EXTRAM: Internal/External RAM access 内部/外部RAM存取

0：内部扩展的EXT_RAM可以存取.
STC15W4K60S4系列单片机
在00H到EFFH单元(3840字节),使用MOVX @DPTR指令访问,超过F00H(含F00H单元)的地址空间总是访问外部数据存储器，MOVX @Ri只能访问00H到FFH单元

1：External data memory access. 外部数据存储器存取
禁止访问内部扩展RAM，此时MOVX @DPTR/MOVX @Ri的使用同普通8052单片机

助记符	功能说明	所需时钟	条件 (以STC15W4K60S4系列为例，即假定片内扩展RAM为3840字节)
MOVX @DPTR, A	将累加器A的内容送入逻辑上在片外、物理上在片内的扩展RAM(16位地址)中，写操作	3	DPTR的内容为0000H ~ 0EFFH 即(3840字节=[4096-256])
MOVX A, @DPTR	将逻辑上在片外、物理上在片内的扩展RAM(16位地址)的内容送入累加器A中，读操作	2	DPTR的内容为0000H ~ 0EFFH 即(3840字节=[4096-256])
MOVX @Ri, A	将累加器A的内容送入逻辑上在片外、物理上在片内的扩展RAM(8位地址)中，写操作	4	EXTRAM=0
MOVX A, @Ri	将逻辑上在片外、物理上在片内的扩展RAM(8位地址)的内容送入累加器A中，读操作	3	EXTRAM=0
MOVX @Ri, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(8位地址)中，写操作	8	EXRTS[1:0] = [0,0], EXTRAM=1
MOVX A, @Ri	将逻辑上在片外、物理上也在片外的扩展RAM(8位地址)的内容送入累加器A中，读操作	7	EXRTS[1:0] = [0,0], EXTRAM=1
MOVX @Ri, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(8位地址)中，写操作	13	EXRTS[1:0] = [0,1], EXTRAM=1
MOVX A, @Ri	将逻辑上在片外、物理上也在片外的扩展RAM(8位地址)的内容送入累加器A中，读操作	12	EXRTS[1:0] = [0,1], EXTRAM=1
MOVX @Ri, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(8位地址)中，写操作	23	EXRTS[1:0] = [1,0], EXTRAM=1

STC MCU Limited

助记符	功能说明	所需时钟	条件 (以STC15W4K60S4系列为例，即假定片内扩展RAM为3840字节)
MOVX A, @Ri	将逻辑上在片外、物理上也在片外的扩展RAM(8位地址)的内容送入累加器A中，读操作	22	EXRTS[1:0] = [1,0], EXTRAM=1
MOVX @Ri, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(8位地址)中，写操作	43	EXRTS[1:0] = [1,1], EXTRAM=1
MOVX A, @Ri	将逻辑上在片外、物理上也在片外的扩展RAM(8位地址)的内容送入累加器A中，读操作	42	EXRTS[1:0] = [1,1], EXTRAM=1
注意：对于传统8051单片机，Ri只能取R0和R1，STC15系列单片机与传统8051单片机一样，Ri也只能为R0或R1，即上表中Ri中的i=0,1.			
助记符	功能说明	所需时钟	条件 (以STC15W4K60S4系列为例，即假定片内扩展RAM为3840字节)
MOVX @DPTR, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(16位地址)中，写操作	7	EXRTS[1:0] = [0,0], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX A, @DPTR	将逻辑上在片外、物理上也在片外的扩展RAM(16位地址)的内容送入累加器A中，读操作	6	EXRTS[1:0] = [0,0], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX @DPTR, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(16位地址)中，写操作	12	EXRTS[1:0] = [0,1], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX A, @DPTR	将逻辑上在片外、物理上也在片外的扩展RAM(16位地址)的内容送入累加器A中，读操作	11	EXRTS[1:0] = [0,1], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX @DPTR, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(16位地址)中，写操作	22	EXRTS[1:0] = [1,0], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX A, @DPTR	将逻辑上在片外、物理上也在片外的扩展RAM(16位地址)的内容送入累加器A中，读操作	21	EXRTS[1:0] = [1,0], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX @DPTR, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(16位地址)中，写操作	42	EXRTS[1:0] = [1,1], DPTR>=3840 即(4096-256) or EXTRAM=1
MOVX A, @DPTR	将逻辑上在片外、物理上也在片外的扩展RAM(16位地址)的内容送入累加器A中，读操作	41	EXRTS[1:0] = [1,1], DPTR>=3840 即(4096-256) or EXTRAM=1

访问片外扩展RAM指令所需时钟计算公式：

MOVX @R0/R1　　MOVX @DPTR

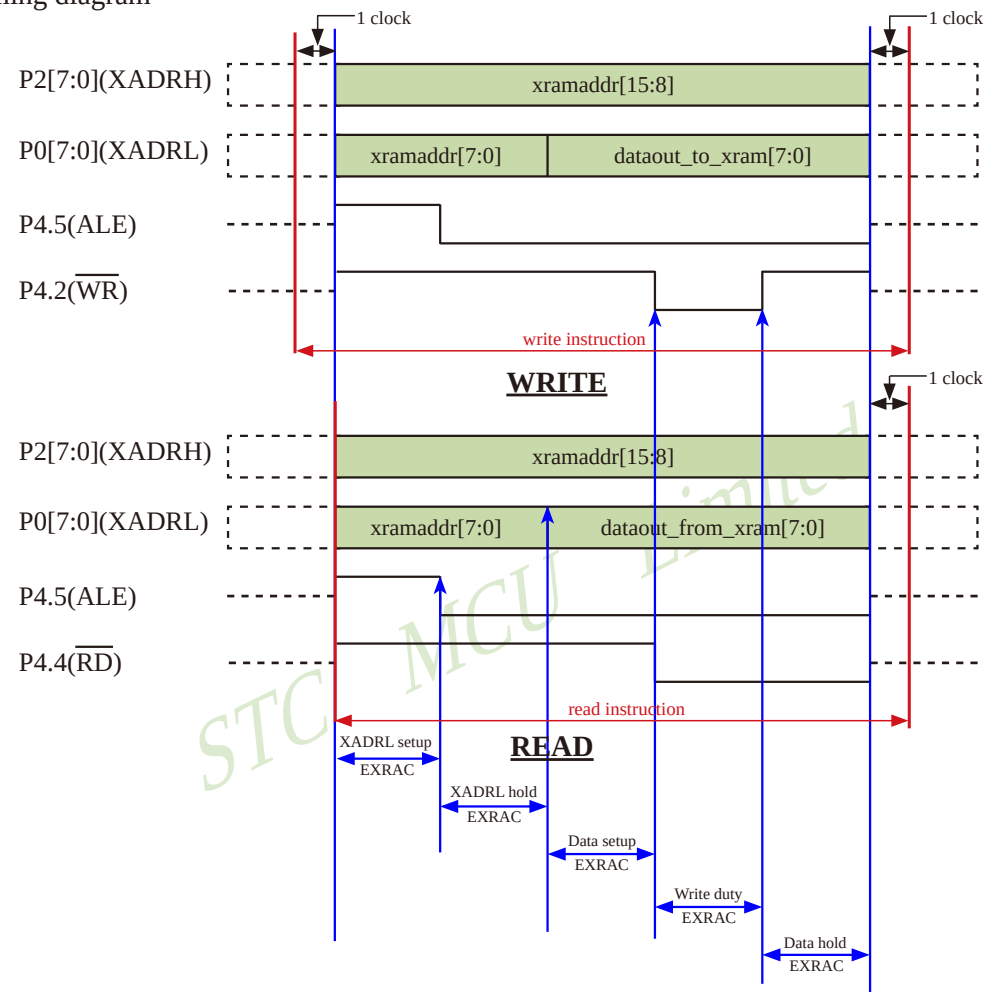
write(写操作): 5×N+3　　write(写操作): 5×N+2

read(读操作): 5×N+2　　read(读操作): 5×N+1

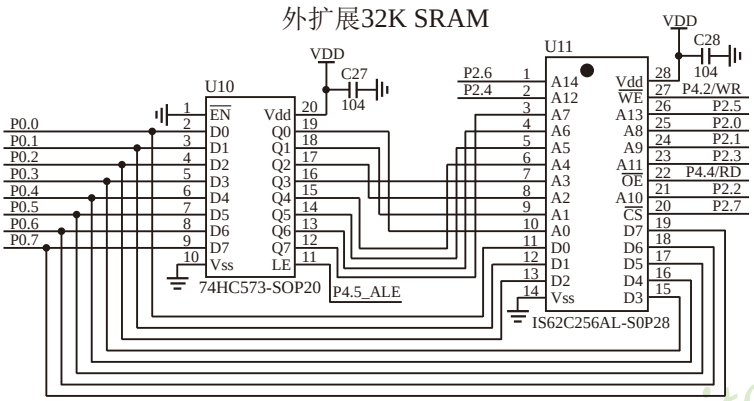
当EXRTS[1:0] = [0,0]时，上式中N=1；
当EXRTS[1:0] = [0,1]时，上式中N=2；
当EXRTS[1:0] = [1,0]时，上式中N=4；
当EXRTS[1:0] = [1,1]时，上式中N=8.

可见，对于STC15系列单片机，访问片外扩展RAM指令的速度是可调的

Timing diagram



3.2.4 利用并行总线扩展外部32K SRAM的应用线路图



注意：由于历史原因，IS62C256AL-S0P28封装比STC的SOP28封装要宽

STC MCU Limited

3.3 特殊功能寄存器(SFRs)

特殊功能寄存器(SFR)是用来对片内各功能模块进行管理、控制、监视的控制寄存器和状态寄存器，是一个特殊功能的RAM区。STC15系列单片机内的特殊功能寄存器(SFR)与高128字节RAM共用相同的地址范围，都使用80H~FFH，特殊功能寄存器(SFR)必须用直接寻址指令访问。请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无耻了。

STC15W4K60S4系列单片机的特殊功能寄存器名称及地址映象如下表所示

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
0F8H	P7 0000,0000	CH 0000,0000	CCAP0H 0000,0000	CCAP1H 0000,0000	CCAP2H 0000,0000				0FFH
0F0H	B 0000,0000		PCA_PWM0 00xx,xx00	PCA_PWM1 00xx,xx00	PCA_PWM2 00xx,xx00				0F7H
0E8H	P6 0000,0000	CL 0000,0000	CCAP0L 0000,0000	CCAP1L 0000,0000	CCAP2L 0000,0000				0EFH
0E0H	ACC 0000,0000	P7M1	P7M0						0E7H
0D8H	CCON 00xx,0000	CMOD 0xxx,x000	CCAPM0 x000,0000	CCAPM1 x000,0000	CCAPM2 x000,0000				0DFH
0D0H	PSW 0000,00x0	T4T3M 0000,0000	T4H RL_TH4 0000,0000	T4L RL_TL4 0000,0000	T3H RL_TH3 0000,0000	T3L RL_TL3 0000,0000	T2H RL_TH2 0000,0000	T2L RL_TL2 0000,0000	0D7H
0C8H	P5 xxxx,1111	P5M1 xxxx,0000	P5M0 xxxx,0000	P6M1	P6M0	SPSTAT 00xx,xxxx	SPCTL 0000,0100	SPDAT 0000,0000	0CFH
0C0H	P4 1111,1111	WDT_CONTR 0x00,0000	IAP_DATA 1111,1111	IAP_ADDRH 0000,0000	IAP_ADDRL 0000,0000	IAP_CMD xxxx,xx00	IAP_TRIG xxxx,xxxx	IAP_CONTR 0000,0000	0C7H
0B8H	IP x0x0,0000	SADEN	P_SW2 xxxx,x000		ADC_CONTR 0000,0000	ADC_RES 0000,0000	ADC_RESL 0000,0000		0BFH
0B0H	P3 1111,1111	P3M1 0000,0000	P3M0 0000,0000	P4M1 0000,0000	P4M0 0000,0000	IP2 xxxx,xx00	IP2H xxxx,xx00	IPH 0000,0000	0B7H
0A8H	IE 0000,0000	SADDR	WKTCL WKTCL_CNT 0111 1111	WKTCH WKTCH_CNT 0111 1111	S3CON 0000,0000	S3BUF xxxx,xxxx		IE2 x000,0000	0AFH
0A0H	P2 1111,1111	BUS_SPEED xxxx,xx10	AUXR1 P_SW1 0100,0000	Don't use	Don't use	Don't use		Don't use	0A7H
098H	SCON 0000,0000	SBUF xxxx,xxxx	S2CON 0000,0000	S2BUF xxxx,xxxx	Don't use	P1ASF 0000,0000	Don't use	Don't use	09FH
090H	P1 1111,1111	P1M1 0000,0000	P1M0 0000,0000	P0M1 0000,0000	P0M0 0000,0000	P2M1 0000,0000	P2M0 0000,0000	CLK_DIV PCON2	097H
088H	TCON 0000,0000	TMOD 0000,0000	TL0 RL_TL0 0000,0000	TL1 RL_TL1 0000,0000	TH0 RL_TH0 0000,0000	TH1 RL_TH1 0000,0000	AUXR 0000,0001	INT_CLKO AUXR2 0000 0000	08FH
080H	P0 1111,1111	SP 0000,0111	DPL 0000,0000	DPH 0000,0000	S4CON 0000,0000	S4BUF xxxx,xxxx		PCON 0011,0000	087H
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

可位寻址　　不可位寻址

注意：寄存器地址能够被8整除的才可以进行位操作，不能够被8整除的不可以进行位操作

STC MCU Limited

符号	描述	地址	位地址及符号 MSB LSB								复位值
P0	Port 0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	1111 1111B
SP	堆栈指针	81H									0000 0111B
DPTR DPL	数据指针(低)	82H									0000 0000B
DPTR DPH	数据指针(高)	83H									0000 0000B
S4CON	串口4控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口4数据缓冲器	85H									xxxx,xxxx
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
TMOD	定时器工作方式寄存器	89H	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0	0000 0000B
TL0	定时器0低8位寄存器	8AH									0000 0000B
TL1	定时器1低8位寄存器	8BH									0000 0000B
TH0	定时器0高8位寄存器	8CH									0000 0000B
TH1	定时器1高8位寄存器	8DH									0000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	外部中断允许和时钟输出寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000 x000B
P1	Port 1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	1111 1111B
P1M1	P1口模式配置寄存器1	91H									0000 0000B
P1M0	P1口模式配置寄存器0	92H									0000 0000B
P0M1	P0口模式配置寄存器1	93H									0000 0000B
P0M0	P0口模式配置寄存器0	94H									0000 0000B
P2M1	P2口模式配置寄存器1	95H									0000 0000B
P2M0	P2口模式配置寄存器0	96H									0000 0000B
CLK_DIV PCON2	时钟分频寄存器	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
SCON	串口1控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口1数据缓冲器	99H									xxxx,xxxx
S2CON	串口2控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000,0000
S2BUF	串口2数据缓冲器	9BH									xxxx,xxxx

STC MCU Limited

符号	描述	地址	位地址及符号 MSB LSB								复位值
P1ASF	P1 Analog Function Configure register	9DH	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF	0000 0000B
P2	Port 2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	1111 1111B
BUS_SPEED	Bus-Speed Control	A1H	-	-	-	-	-	-	EXRTS[1:0]		xxxx xx10B
AUXR1 P_SW1	辅助寄存器1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000B
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
SADDR	从机地址控制寄存器	A9H									0000 0000B
WKTCL WKTCL_CNT	掉电唤醒专用定时器控制寄存器低8位	AAH									1111 1111B
WKTCH WKTCH_CNT	掉电唤醒专用定时器控制寄存器高8位	ABH	WKTEN								0111 1111B
S3CON	串口3控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口3数据缓冲器	ADH									xxxx,xxxx
IE2	中断允许寄存器	AFH		ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
P3	Port 3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	1111 1111B
P3M1	P3口模式配置寄存器1	B1H									0000 0000B
P3M0	P3口模式配置寄存器0	B2H									0000 0000B
P4M1	P4口模式配置寄存器1	B3H									0000 0000B
P4M0	P4口模式配置寄存器0	B4H									0000 0000B
IP2	第二中断优先级低字节寄存器	B5H	-	-	-	-	-	-	PSPI	PS2	xxxx xx00B
IP	中断优先级寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
SADEN	从机地址掩模寄存器	B9H									0000 0000B
P_SW2	外围设备功能切换控制寄存器	BAH	-	-	-	-	-	S4_S	S3_S	S2_S	xxxx x000B
ADC_CONTR	A/D转换控制寄存器	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0	0000 0000B
ADC_RES	A/D转换结果高8位寄存器	BDH									0000 0000B
ADC_RESL	A/D转换结果低2位寄存器	BEH									0000 0000B
P4	Port 4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0	1111 1111B

STC MCU Limited

符号	描述	地址	位地址及符号 MSB LSB								复位值
WDT_CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0	0x00 0000B
IAP_DATA	ISP/IAP 数据寄存器	C2H									1111 1111B
IAP_ADDRH	ISP/IAP 高8位地址寄存器	C3H									0000 0000B
IAP_ADDRL	ISP/IAP 低8位地址寄存器	C4H									0000 0000B
IAP_CMD	ISP/IAP 命令寄存器	C5H	-	-	-	-	-	-	MS1	MS0	xxxx xx00B
IAP_TRIG	ISP/IAP 命令触发寄存器	C6H									xxxx xxxxB
IAP_CONTR	ISP/IAP控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0	0000 x000B
P5	Port 5	C8H	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0	xx11 1111B
P5M1	P5口模式配置寄存器1	C9H									xxx0 0000B
P5M0	P5口模式配置寄存器0	CAH									xxx0 0000B
P6M1	P6口模式配置寄存器1	CBH									
P6M0	P6口模式配置寄存器0	CCH									
SPSTAT	SPI状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx xxxxB
SPCTL	SPI控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CAPHA	SPR1	SPR0	0000 0100B
SPDAT	SPI数据寄存器	CFH									0000 0000B
PSW	程序状态字寄存器	D0H	CY	AC	F0	RS1	RS0	OV	-	P	0000 00x0B
T4T3M	T4和T3的控制寄存器	D1H	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO	0000 0000B
T4H	定时器4高8位寄存器	D2H									0000 0000B
T4L	定时器4低8位寄存器	D3H									0000 0000B
T3H	定时器3高8位寄存器	D4H									0000 0000B
T3L	定时器3低8位寄存器	D5H									0000 0000B
T2H	定时器2高8位寄存器	D6H									0000 0000B
T2L	定时器2低8位寄存器	D7H									0000 0000B
CCON	PCA控制寄存器	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx 0000B
CMOD	PCA模式寄存器	D9H	CIDL	-	-	-	-	CPS1	CPS0	ECF	0xxx x000B
CCAPM0	PCA Module 0 Mode Register	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x000 0000B
CCAPM1	PCA Module 1 Mode Register	DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x000 0000B
CCAPM2	PCA Module 2 Mode Register	DCH	-	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	x000 0000B

STC MCU Limited

符号	描述	地址	位地址及符号 MSB LSB								复位值
ACC	累加器	E0H									0000 0000B
P7M1	P7口模式配置寄存器1	E1H									
P7M0	P7口模式配置寄存器0	E2H									
P6	Port 6	E8H									
CL	PCA Base Timer Low	E9H									0000 0000B
CCAP0L	PCA Module-0 Capture Register Low	EAH									0000 0000B
CCAP1L	PCA Module-1 Capture Register Low	EBH									0000 0000B
CCAP2L	PCA Module-2 Capture Register Low	ECH									0000 0000B
B	B寄存器	F0H									0000 0000B
PCA_PWM0	PCA PWM Mode Auxiliary Register 0	F2H	EBS0_1	EBS0_0	-	-	-	-	EPC0H	EPC0L	xxxx xx00B
PCA_PWM1	PCA PWM Mode Auxiliary Register 1	F3H	EBS1_1	EBS1_0	-	-	-	-	EPC1H	EPC1L	xxxx xx00B
PCA_PWM2	PCA PWM Mode Auxiliary Register 2	F4H	EBS2_1	EBS2_0	-	-	-	-	EPC2H	EPC2L	xxxx xx00B
P7	Port 7	F8H									
CH	PCA Base Timer High	F9H									0000 0000B
CCAP0H	PCA Module-0 Capture Register High	FAH									0000 0000B
CCAP1H	PCA Module-1 Capture Register High	FBH									0000 0000B
CCAP2H	PCA Module-2 Capture Register High	FCH									0000 0000B

STC MCU Limited

下面简单的介绍一下普通8051单片机常用的一些寄存器：

1. 程序计数器(PC)

程序计数器PC在物理上是独立的，不属于SFR之列。PC字长16位，是专门用来控制指令执行顺序的寄存器。单片机上电或复位后，PC=0000H，强制单片机从程序的零单元开始执行程序。

2. 累加器(ACC)

累加器ACC是8051单片机内部最常用的寄存器，也可写作A。常用于存放参加算术或逻辑运算的操作数及运算结果。

3. B寄存器

B寄存器在乘法和除法运算中须与累加器A配合使用。MUL AB指令把累加器A和寄存器B中的8位无符号数相乘，所得的16位乘积的低字节存放在A中，高字节存放在B中。DIV AB指令用B除以A，整数商存放在A中，余数存放在B中。寄存器B还可以用作通用暂存寄存器。

4. 程序状态字(PSW)寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	name	CY	AC	F0	RS1	RS0	OV	-	P

CY ： 标志位。进行加法运算时，当最高位即B7位有进位，或执行减法运算最高位有借位时，CY为1；反之为0

AC ： 进位辅助位。进行加法运算时，当B3位有进位，或执行减法运算B3有借位时，AC为1；反之为0。设置辅助进位标志AC的目的是为了便于BCD码加法、减法运算的调整。

F0 ： 用户标志位。

RS1、RS0: 工作寄存器组的选择位。RS1、RS0: 工作寄存器组的选择位。如下表

RS1	RS0	当前使用的工作寄存器组(R0~R7)
0	0	0组(00H~07H)
0	1	1组(08H~0FH)
1	0	2组(10H~17H)
1	1	3组(18H~1FH)

OV：溢出标志位.

B1 ：保留位

P ： 奇偶标志位。该标志位始终体现累加器ACC中1的个数的奇偶性。如果累加器ACC中1的个数为奇数，则P置1；当累加器ACC中的个数为偶数(包括0个)时，P位为0

STC MCU Limited

5. 堆栈指针(SP)

堆栈指针是一个8位专用寄存器。它指示出堆栈顶部在内部RAM块中的位置。系统复位后，SP初始化位07H，使得堆栈事实上由08H单元开始，考虑08H~1FH单元分别属于工作寄存器组1~3，若在程序设计中用到这些区，则最好把SP值改变为80H或更大的值为宜。STC15系列单片机的堆栈是向上生长的，即将数据压入堆栈后，SP内容增大。

6. 数据指针(DPTR)

数据指针(DPTR)是一个16位专用寄存器，由DPL(低8位)和DPH(高8位)组成,地址是82H(DPL,低字节)和83H(DPH,高字节)。DPTR是传统8051机中唯一可以直接进行16位操作的寄存器也可分别对DPL河DPH按字节进行操作。

如果用户所使用的STC15系列单片机无外部数据总线，那么该单片机只设计了一个16位的数据指针。相反，如果用户所使用的STC15系列单片机有外部数据总线，那么该单片机设计了两个16位的数据指针DPRT0和DPTR1，这两个数据指针共用同一个地址空间，可通过设置DPS/P_SW1.0来选择具体被使用的数据指针。

当用户所使用的STC15系列单片机有外部数据总线，即该单片机设计了两个16位的数据指针DPRT0和DPTR1时，这两个数据指针可通过软件设置DPS/P_SW1.0来选择具体哪个数据指针被使用，见下面寄存器AUXR1的说明。

STC15系列8051 单片机 双数据指针 特殊功能寄存器

Mnemonic	Address	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary Register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,0000

DPS　DPTR registers select bit. DPTR 寄存器选择位

0: DPTR0 is selected　DPTR0被选择

1: DPTR1 is selected　DPTR1被选择

此系列单片机有两个16-bit数据指针,DPTR0,DPTR1.当DPS选择位为0时,选择DPTR0,当DPS选择位为1时,选择DPTR1.

AUXR1特殊功能寄存器,位于A2H单元,其中的位不可用布尔指令快速访问.但由于DPS位位于bit0,故对AUXR1寄存器用INC指令,DPS位便会反转,由0变成1或由1变成0,即可实现双数据指针的快速切换.

STC MCU Limited

应用示例供参考:

```
;新增特殊功能寄存器定义
    P_SW1       DATA    0A2H
    MOV         P_SW1,  #0              ;此时DPS为0,DPTR0有效

    MOV         DPTR,   #1FFH           ;置DPTR0为1FFH
    MOV         A,      #55H
    MOVX        @DPTR,  A               ;将1FFH单元置为55H

    MOV         DPTR,   #2FFH           ;置DPTR0为2FFH
    MOV         A,      #0AAH
    MOVX        @DPTR,  A               ;将2FFH单元置为0AAH

    INC         P_SW1                   ; 此时DPS为1,DPTR1有效
    MOV         DPTR,   #1FFH           ; 置DPTR1为1FFH
    MOVX        A,      @DPTR   ;读DPTR1数据指针指向的1FFH单元的内容,累加器A变为55H.

    INC         P_SW1                   ; 此时DPS为0,DPTR0有效
    MOVX        A,      @DPTR   ;读DPTR0数据指针指向的2FFH单元的内容,累加器A变为0AAH.

    INC         P_SW1                   ; 此时DPS为1,DPTR1有效
    MOVX        A,      @DPTR   ; 读DPTR1数据指针指向的1FFH单元的内容,累加器A变为55H.

    INC         P_SW1                   ; 此时DPS为0,DPTR0有效
    MOVX        A,      @DPTR   ;读DPTR0数据指针指向的2FFH单元的内容,累加器A变为0AAH.
```

当用户所使用的STC系列单片机无外部数据总线，即该单片机只设计了一个16位的数据指针时，DPS/P_SW1.0位无效。

Mnemonic	Address	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary Register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,000x

STC MCU Limited

第4章 STC15系列单片机的I/O口结构

4.1 I/O口各种不同的工作模式及配置介绍

I/O口配置

STC15系列单片机最多有46个I/O口(如48-pin单片机)：P0.0~P0.7, P1.0~P1.7, P2.0~P2.7, P3.0~P3.7, P4.0~P4.7, P5.0~P5.5。其所有I/O口均可由软件配置成4种工作类型之一，如下表所示。4种类型分别为：准双向口/弱上拉（标准8051输出模式）、推挽输出/强上拉、仅为输入（高阻）或开漏输出功能。每个口由2个控制寄存器中的相应位控制每个引脚工作类型。STC15系列单片机的I/O口上电复位后为准双向口/弱上拉（传统8051的I/O口）模式。每个I/O口驱动能力均可达到20mA，但40-pin及40-pin以上单片机的整个芯片最大不要超过120mA，20-pin以上及32-pin以下(包括32-pin)单片机的整个芯片最大不要超过90mA。

I/O口工作类型设定

P5口设定 〈X, X, P5.5, P5.4, P5.3, P5.2, P5.1, P5.0口〉(P5口地址：C8H)

P5M1[7 : 0] 寄存器P5M1地址为C9H	P5M0[7 : 0] 寄存器P5M0地址为CAH	I/O 口模式
0	0	准双向口(传统8051 I/O 口模式)， 灌电流可达20mA, 拉电流为270μA, 由于制造误差，实际为270uA～150uA
0	1	推挽输出(强上拉输出，可达20mA, 要加限流电阻)
1	0	仅为输入(高阻)
1	1	开漏(Open Drain)，内部上拉电阻断开。 开漏模式既可读外部状态也可对外输出(高电平或低电平)。 如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。

举例：　MOV　P5M1,　#00101000B
　　MOV　P5M0,　#00110000B
;P5.5为开漏,P5.4为强推挽输出,P5.3为高阻输入, P5.2/P5.1/P5.0为准双向口/弱上拉

P4口设定 〈P4.7, P4.6, P4.5, P4.4, P4.3, P4.2, P4.1, P4.0〉(P4口地址：C0H)

P4M1[7 : 0] 寄存器P4M1地址为B3H	P4M0[7 : 0] 寄存器P4M0地址为B4H	I/O 口模式
0	0	准双向口(传统8051 I/O 口模式)， 灌电流可达20mA, 拉电流为270μA, 由于制造误差，实际为270uA～150uA
0	1	推挽输出(强上拉输出，可达20mA, 要加限流电阻)
1	0	仅为输入(高阻)
1	1	开漏(Open Drain)，内部上拉电阻断开。 开漏模式既可读外部状态也可对外输出(高电平或低电平)。 如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。

举例：　MOV　P4M1,　#10100000B
　　MOV　P4M0,　#11000000B
;P4.7为开漏,P4.6为强推挽输出,P4.5为高阻输入,P4.4/P4.3/P4.2/P4.1/P4.0为准双向口/弱上拉

P3口设定〈P3.7，P3.6，P3.5，P3.4，P3.3，P3.2，P3.1，P3.0口〉(P3口地址：B0H)

P3M1[7 : 0] 寄存器P3M1地址为B1H	P3M0[7 : 0] 寄存器P3M0地址为B2H	I/O 口模式
0	0	准双向口(传统8051 I/O 口模式)， 灌电流可达20mA, 拉电流为270μA, 由于制造误差，实际为270uA～150uA
0	1	推挽输出(强上拉输出，可达20mA, 要加限流电阻)
1	0	仅为输入(高阻)
1	1	开漏(Open Drain)，内部上拉电阻断开。 开漏模式既可读外部状态也可对外输出(高电平或低电平)。 如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。

```
举例:   MOV     P3M1,   #10100000B
        MOV     P3M0,   #11000000B
  ;P3.7为开漏,P3.6为强推挽输出,P3.5为高阻输入,P3.4/P3.3/P3.2/P3.1/P3.0为准双向口/弱上拉
```

P2口设定〈P2.7，P2.6，P2.5，P2.4，P2.3，P2.2，P2.1，P2.0〉(P2口地址：A0H)

P2M1[7 : 0] 寄存器P2M1地址为95H	P2M0[7 : 0] 寄存器P2M0地址为96H	I/O 口模式
0	0	准双向口(传统8051 I/O 口模式)， 灌电流可达20mA, 拉电流为270μA, 由于制造误差，实际为270uA～150uA
0	1	推挽输出(强上拉输出，可达20mA, 要加限流电阻)
1	0	仅为输入(高阻)
1	1	开漏(Open Drain)，内部上拉电阻断开。 开漏模式既可读外部状态也可对外输出(高电平或低电平)。 如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。

```
举例:   MOV     P2M1,   #10100000B
        MOV     P2M0,   #11000000B
  ;P2.7为开漏,P2.6为强推挽输出,P2.5为高阻输入,P2.4/P2.3/P2.2/P2.1/P2.0为准双向口/弱上拉
```

P1口设定〈P1.7，P1.6，P1.5，P1.4，P1.3，P1.2，P1.1，P1.0口〉(P1口地址：90H)

P1M1[7 : 0] 寄存器P1M1地址为91H	P1M0[7 : 0] 寄存器P1M0地址为92H	I/O 口模式 （P1.x 如做A/D使用，需先将其设置成开漏或高阻输入）
0	0	准双向口(传统8051 I/O 口模式)， 灌电流可达20mA, 拉电流为270μA, 由于制造误差，实际为270uA～150uA
0	1	推挽输出(强上拉输出，可达20mA, 要加限流电阻)
1	0	仅为输入(高阻)
1	1	开漏(Open Drain)，内部上拉电阻断开。 开漏模式既可读外部状态也可对外输出(高电平或低电平)。 如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。

```
举例:   MOV     P1M1,   #10100000B
        MOV     P1M0,   #11000000B
  ;P1.7为开漏,P1.6为强推挽输出,P1.5为高阻输入,P1.4/P1.3/P1.2/P1.1/P1.0为准双向口/弱上拉
```

STC MCU Limited

P0口设定 < P0.7, P0.6, P0.5, P0.4, P0.3, P0.2, P0.1, P0.0口>(P0口地址：80H)

P0M1 [1：0] 寄存器P0M1地址为93H	P0M0 [1：0] 寄存器P0M0地址为94H	I/O 口模式
0	0	准双向口(传统8051 I/O 口模式), 灌电流可达20mA, 拉电流为270μA, 由于制造误差，实际为270uA～150uA
0	1	推挽输出(强上拉输出，可达20mA, 要加限流电阻)
1	0	仅为输入(高阻)
1	1	开漏(Open Drain)，内部上拉电阻断开。 开漏模式既可读外部状态也可对外输出(高电平或低电平)。 如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。

举例：　MOV　P0M1, #10100000B
　　MOV　P0M0, #11000000B
;P0.7为开漏,P0.6为强推挽输出,P0.5为高阻输入,P0.4/P0.3/P0.2/P0.1/P0.0为准双向口/弱上拉

注意:

虽然每个I/O口在弱上拉/强推挽输出/开漏模式时都能承受20mA的灌电流(还是要加限流电阻,如1K，560Ω，472Ω等),在强推挽输出时能输出20mA的拉电流（也要加限流电阻）,但整个芯片的工作电流推荐不要超过90mA，**即从MCU-VCC流入的电流建议不要超过90mA,从MCU-Gnd流出电流建议不要超过90mA,整体流入/流出电流建议都不要超过90mA.**

STC MCU Limited

4.2 管脚P1.7/XTAL1与P1.6/XTAL2的特别说明

STC15系列单片机的所有I/O口上电复位后均为准双向口/弱上拉模式。但是由于P1.7和P1.6口还可以分别作外部晶体或时钟电路的引脚XTAL1和XTAL2，所以P1.7/XTAL1和P1.6/XTAL2上电复位后的模式不一定就是准双向口/弱上拉模式。当P1.7和P1.6口作为外部晶体或时钟电路的引脚XTAL1和XTAL2使用时，P1.7/XTAL1和P1.6/XTAL2上电复位后的模式是高阻输入。

每次上电复位时，单片机对P1.7/XTAL1和P1.6/XTAL2的工作模式按如下步骤进行设置：

1. 首先，单片机短时间（几十个时钟）内会将P1.7/XTAL1和P1.6/XTAL2设置成高阻输入；
2. 然后，单片机会自动判断上一次用户ISP烧录程序时是将P1.7/XTAL1和P1.6/XTAL2设置成普通I/O口还是XTAL1/XTAL2；
3. 如果上一次用户ISP烧录程序时是将P1.7/XTAL1和P1.6/XTAL2设置成普通I/O口,则单片机会将P1.7/XTAL1和P1.6/XTAL2上电复位后的模式设置成准双向口/弱上拉；
4. 如果上一次用户ISP编程时是将P1.7/XTAL1和P1.6/XTAL2设置成XTAL1/XTAL2,则单片机会将P1.7/XTAL1和P1.6/XTAL2上电复位后的模式设置成高阻输入。

4.3 复位管脚RST的特别说明

STC15系列8-pin单片机(如STC15F101W系列)的复位管脚在RST/P3.4口，STC15系列16-pin及其以上的单片机(如STC15W4K60S4系列、STC15F2K60S2系列等)的复位管脚在RST/P5.4口。下面以RST/P5.4为例，介绍复位管脚RST。

P5.4/RST(或P3.4/RST)即可作普通I/O使用，还可作复位管脚，用户可以在ISP烧录程序时设置P5.4/RST的功能。当用户ISP烧录程序时将P5.4/RST设置成普通I/O口用时，其上电后为准双向口/弱上拉模式。

每次上电时，单片机会自动判断上一次用户ISP烧录程序时是将P5.4/RST设置成普通I/O口还是复位脚。如果上一次用户在ISP编程时是将P5.4/RST设置成普通I/O口,则单片机会将P5.4/RST上电后的模式设置成准双向口/弱上拉。如果上一次用户ISP烧录程序时是将P5.4/RST设置成复位脚,则上电后，P5.4/RST仍为复位脚。

4.4 管脚RSTOUT_LOW的特别说明

STC15系列有的单片机(如STC15W4K60S4及STC15F2K60S2系列)的RSTOUT_LOW脚在P2.0口(P2.0/RSTOUT_LOW)，有的单片机(如STC15W104SW系列)的RSTOUT_LOW脚在P1.0口(P1.0/RSTOUT_LOW)，有的单片机(如STC15F101W系列)的RSTOUT_LOW脚在P3.3口(P3.3/RSTOUT_LOW)。下面以P2.0/RSTOUT_LOW管脚为例，介绍管脚P2.0/RSTOUT_LOW的一些特别注意事项。

P2.0/RSTOUT_LOW管脚在单片机上电复位后输出可以为低电平，也可以为高电平。当单片机的工作电压Vcc高于上电复位门槛电压(POR)时，用户可以在ISP烧录程序时设置该管脚上电复位后输出的是低电平还是高电平。

当单片机的工作电压Vcc低于上电复位门槛电压(POR，3V单片机在1.8V附近，5V单片机在3.2V附近)时，P2.0/RSTOUT_LOW管脚输出低电平。当单片机的工作电压Vcc高于上电复位门槛电压(POR，3V单片机在1.8V附近，5V单片机在3.2V附近)时，单片机首先读取用户在ISP烧录程序时的设置，如用户将P2.0/RSTOUT_LOW管脚设置为上电复位后输出高电平，则P2.0/RSTOUT_LOW管脚上电复位后输出高电平；如用户将P2.0/RSTOUT_LOW管脚设置为上电复位后输出低电平，则P2.0/RSTOUT_LOW管脚上电复位后输出低电平。

STC MCU Limited

4.5 串行口1的中继广播方式

串行口1可在3组管脚间进行切换：[RxD/P3.0, TxD/P3.1]；
[RxD_2/P3.6, TxD_2/P3.7]；
[RxD_3/P1.6, TxD_3/P1.7].

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,x000

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

串口1的RxD管脚和TxD管脚可以在3组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1]；
[RxD_2/P3.6, TxD_2/P3.7]；
[RxD_3/P1.6, TxD_3/P1.7].

串行口1的中继广播方式除可以在用户程序中设置Tx_Rx/CLK_DIV.4来选择外，还可以在STC-ISP下载编程软件中设置。

当单片机的工作电压低于上电复位门槛电压(POR，3V单片机在1.8V附近，5V单片机在3.2V附近)时，Tx_Rx默认为0，即串行口1默认为正常工作方式。当单片机的工作电压高于上电复位门槛电压(POR，3V单片机在1.8V附近，5V单片机在3.2V附近)时，单片机首先读取用户在STC-ISP下载编程软件中的设置，如果用户允许了“单片机TxD管脚的对外输出实时反映RxD端口输入的电平状态”即中继广播方式，则上电复位后P3.7/TxD_2管脚的对外输出可以实时反映P3.6/RxD_2端口输入的电平状态，如果用户未选择“单片机TxD管脚的对外输出实时反映RxD端口输入的电平状态”，则上电复位后串口1为正常工作方式，即P3.7/TxD_2管脚的对外输出不实时反映P3.6/RxD_2端口输入的电平状态。

串行口1的位置和中继广播方式除可以在STC-ISP下载编程软件中设置外，还可以在用户的用户程序中用设置。在STC-ISP下载编程软件中的设置是在单片机上电复位后就可以执行的，如果用户在用户程序中的设置与STC-ISP下载编程软件中的设置不一致时，当执行到相应的用户程序时就会覆盖原来STC-ISP下载编程软件中的设置。

4.6 可将MCU从掉电模式/停机模式唤醒的外部管脚资源

可将MCU从掉电模式/停机模式唤醒的外部管脚资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚CCP0/CCP1/CCP2；管脚RxD/RxD2/RxD3/RxD4(下降沿，不产生中断)；管脚T0/T1/T2/T3/T4(下降沿即外部管脚由高到低的变化，前提是在进入掉电模式/停机模式前相应的定时器中断已经被允许)；内部低功耗掉电唤醒专用定时器。

INT0/P3.2和INT1/P3.3的上升沿/下降沿中断均可唤醒掉电模式/停机模式。而$\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0仅下降沿中断才可将MCU从掉电模式/停机模式唤醒。

管脚CCP可以在[CCP0/P1.1, CCP1/P1.0, CCP2/CCP2_2/P3.7], [CCP0_2/P3.5, CCP1_2/P3.6, CCP2/CCP2_2/P3.7], [CCP0_3/P2.5, CCP1_3/P2.6, CCP2_3/P2.7]之间进行切换。管脚CCP同样可以将MCU从掉电模式/停机模式中唤醒。

如果掉电模式/停机模式是由外部中断INT0(上升沿+下降沿中断)、INT1(上升沿+下降沿中断)、$\overline{\text{INT2}}$(仅可下降沿中断)、$\overline{\text{INT3}}$(仅可下降沿中断)、$\overline{\text{INT4}}$(仅可下降沿中断)或管脚CCP唤醒，则掉电唤醒之后CPU首先执行设置单片机进入掉电模式的语句的下一条语句（建议在设置单片机进入掉电模式的语句后多加几个NOP空指令），然后执行相应的中断服务程序。

如果在进入掉电模式/停机模式前串行中断被允许，则进入掉电模式/停机模式后，串行口的接收管脚由高到低的变化可以唤醒掉电模式/停机模式。串行口1的接收管脚RxD可以从RxD/P3.0切换到RxD_2/P3.6，还可以切换到RxD_3/P2.6之间切换，上电复位后RxD默认在RxD_2/P3.6管脚上。串行口2的接收管脚RxD2可以从RxD2/P1.0切换到RxD2_2/P4.6，上电复位后RxD2默认在RxD2/P1.0管脚上。串行口3的接收管脚RxD3可以从RxD3/P0.0切换到RxD3_2/P5.0，上电复位后RxD3默认在RxD3/P0.0管脚上。串行口4的接收管脚RxD4可以从RxD4/P0.2切换到RxD4_2/P5.2，上电复位后RxD4默认在RxD4/P0.2管脚上。

如果在进入掉电模式/停机模式前定时器中断被允许，即进入掉电模式/停机模式前ET0/ET1/ET2/ET3/ET4及EA已经被设置为1，则进入掉电模式/停机模式后，定时器的外部管脚(T0/P3.4, T1/P3.5, T2/P3.1, T3/P0.5, T4/P0.7)由高到低的变化也可以将MCU从掉电模式/停机模式唤醒。

当MCU由RxD或RxD2或RxD3或RxD4管脚的下降沿(由高到低的变化)唤醒或由定时器T0/T1/T2/T3/T4的外部管脚的下降沿(由高到低的变化)唤醒时，如果主时钟使用的是内部系统时钟(由用户在ISP烧录程序时自行设置)，**MCU在等待64个时钟后，就认为此时系统时钟从开始起振**的不稳定状态已经过渡到稳定状态,**就将时钟供给CPU工作；如果主时钟使用的是外部晶体或时钟(由用户在ISP烧录程序时自行设置)，MCU在等待1024个时钟后，就认为此时系统时钟从开始**起振的不稳定状态已经过渡到稳定状态,**就将时钟供给CPU工作；CPU获得时钟后，程序从上次**设置单片机进入掉电模式语句的下一条语句开始往下执行。

4.7 与I/O口有关的特殊功能寄存器及其在程序中的地址声明

下面将与I/O口相关的寄存器及其地址列于此处，以方便用户查询

P5 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P5	C8H	name	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0

P5M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P5M1	C9H	name	-	-	P5M1.5	P5M1.4	P5M1.3	P5M1.2	P5M1.1	P5M1.0

P5M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P5M0	CAH	name	-	-	P5M0.5	P5M0.4	P5M0.3	P5M0.2	P5M0.1	P5M0.0

P4 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P4	C0H	name	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0

P4M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P4M1	B3H	name	P4M1.7	P4M1.6	P4M1.5	P4M1.4	P4M1.3	P4M1.2	P4M1.1	P4M1.0

P4M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P4M0	B4H	name	P4M0.7	P4M0.6	P4M0.5	P4M0.4	P4M0.3	P4M0.2	P4M0.1	P4M0.0

P3 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P3	B0H	name	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0

P3M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P3M1	B1H	name	P3M1.7	P3M1.6	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0

P3M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P3M0	B2H	name	P3M0.7	P3M0.6	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0

STC MCU Limited

P2 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P2	A0H	name	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0

P2M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P2M1	95H	name	P2M1.7	P2M1.6	P2M1.5	P2M1.4	P2M1.3	P2M1.2	P2M1.1	P2M1.0

P2M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P2M0	96H	name	P2M0.7	P2M0.6	P2M0.5	P2M0.4	P2M0.3	P2M0.2	P2M0.1	P2M0.0

P1 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1	90H	name	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

P1M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1M1	91H	name	P1M1.7	P1M1.6	P1M1.5	P1M1.4	P1M1.3	P1M1.2	P1M1.1	P1M1.0

P1M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1M0	92H	name	P1M0.7	P1M0.6	P1M0.5	P1M0.4	P1M0.3	P1M0.2	P1M0.1	P1M0.0

P0 register (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	name	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

P0M1 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P0M1	93H	name	P0M1.7	P0M1.6	P0M1.5	P0M1.4	P0M1.3	P0M1.2	P0M1.1	P0M1.0

P0M0 register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	name	P0M0.7	P0M0.6	P0M0.5	P0M0.4	P0M0.3	P0M0.2	P0M0.1	P0M0.0

STC MCU Limited

汇编语言:

```
        P5      EQU     0C8H            ; or  P5        DATA    0C8H
        P5M1    EQU     0C9H            ; or  P5M1      DATA    0C9H
        P5M0    EQU     0CAH
;以上为P5口新增功能寄存器的地址声明
        P4      EQU     0C0H            ; or  P4        DATA    0C0H
        P4M1    EQU     0B3H            ; or  P4M1      DATA    0B3H
        P4M0    EQU     0B4H
;以上为P4口新增功能寄存器的地址声明
        P3M1    EQU     0B1H            ; or  P3M1      DATA    0B1H
        P3M0    EQU     0B2H
;以上为P3口新增功能寄存器的地址声明
        P2M1    EQU     095H
        P2M0    EQU     096H
;以上为P2口新增功能寄存器的地址声明
        P1M1    EQU     091H
        P1M0    EQU     092H
;以上为P1口新增功能寄存器的地址声明
        P0M1    EQU     093H
        P0M0    EQU     094H
;以上为P0口新增功能寄存器的地址声明
```

C语言:

```
        sfr     P5      = 0xc8;
        sfr     P5M1    = 0xc9;
        sfr     P5M0    = 0xca;
/*以上为P5新增功能寄存器的C语言地址声明*/
        sfr     P4      = 0xc0;
        sfr     P4M1    = 0xb3;
        sfr     P4M0    = 0xb4;
/*以上为P4新增功能寄存器的C语言地址声明*/
        sfr     P3M1    = 0xb1;
        sfr     P3M0    = 0xb2;
/*以上为P3新增功能寄存器的C语言地址声明*/
        sfr     P2M1    = 0x95;
        sfr     P2M0    = 0x96;
/*以上为P2新增功能寄存器的C语言地址声明*/
        sfr     P1M1    = 0x91;
        sfr     P1M0    = 0x92;
/*以上为P1新增功能寄存器的C语言地址声明*/
        sfr     P0M1    = 0x93;
        sfr     P0M0    = 0x94;
/*以上为P0新增功能寄存器的C语言地址声明*/
```

STC MCU Limited

4.8 STC15系列单片机P0/P1/P2/P3/P4/P5口的测试程序

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 P0/P1/P2/P3/P4/P5举例-----------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

sfr P5      = 0xC8;          //6 bit Port5      P5.7  P5.6  P5.5  P5.4  P5.3  P5.2  P5.1  P5.0      xxxx1,1111
sfr P5M0 = 0xC9;             //                                                                     0000,0000
sfr P5M1 = 0xCA;             //                                                                     0000,0000
                             //                  7     6     5     4     3     2     1     0       Reset Value

sfr P4      = 0xC0;          //8 bitPort4       P4.7  P4.6  P4.5  P4.4  P4.3  P4.2  P4.1  P4.0      1111,1111
sfr P4M0 = 0xB4;             //                                                                     0000,0000
sfr P4M1 = 0xB3;             //                                                                     0000,0000

sbit    P10     =    P1^0;
sbit    P11     =    P1^1;
sbit    P12     =    P1^2;
sbit    P13     =    P1^3;
sbit    P14     =    P1^4;
sbit    P15     =    P1^5;
sbit    P16     =    P1^6;
sbit    P17     =    P1^7;

sbit    P30     =    P3^0;
sbit    P31     =    P3^1;
sbit    P32     =    P3^2;
sbit    P33     =    P3^3;
sbit    P34     =    P3^4;
sbit    P35     =    P3^5;
sbit    P36     =    P3^6;
sbit    P37     =    P3^7;

sbit    P20     =    P2^0;
sbit    P21     =    P2^1;
sbit    P22     =    P2^2;
```

STC MCU Limited

```
sbit    P23     =       P2^3;
sbit    P24     =       P2^4;
sbit    P25     =       P2^5;
sbit    P26     =       P2^6;
sbit    P27     =       P2^7;

sbit    P00     =       P0^0;
sbit    P01     =       P0^1;
sbit    P02     =       P0^2;
sbit    P03     =       P0^3;
sbit    P04     =       P0^4;
sbit    P05     =       P0^5;
sbit    P06     =       P0^6;
sbit    P07     =       P0^7;

sbit    P40     =       P4^0;
sbit    P41     =       P4^1;
sbit    P42     =       P4^2;
sbit    P43     =       P4^3;
sbit    P44     =       P4^4;
sbit    P45     =       P4^5;
sbit    P46     =       P4^6;
sbit    P47     =       P4^7;

sbit    P50     =       P5^0;
sbit    P51     =       P5^1;
sbit    P52     =       P5^2;
sbit    P53     =       P5^3;
sbit    P54     =       P5^4;
sbit    P55     =       P5^5;


void delay(void);

void main(void)
{
        P10     =       0;
        delay();
        P11     =       0;
        delay();
        P12     =       0;
        delay();
        P13     =       0;
        delay();
        P14     =       0;
        delay();
```

STC MCU Limited

```
    P15     =       0;
    delay();
    P16     =       0;
    delay();
    P17     =       0;
    delay();

    P1      =       0xff;

    P30     =       0;
    delay();
    P31     =       0;
    delay();
    P32     =       0;
    delay();
    P33     =       0;
    delay();
    P34     =       0;
    delay();
    P35     =       0;
    delay();
    P36     =       0;
    delay();
    P37     =       0;
    delay();

    P3      =       0xff;

    P20     =       0;
    delay();
    P21     =       0;
    delay();
    P22     =       0;
    delay();
    P23     =       0;
    delay();
    P24     =       0;
    delay();
    P25     =       0;
    delay();
    P26     =       0;
    delay();
    P27     =       0;
    delay();

    P2      =       0xff;

    P07     =       0;
    delay();
```

STC MCU Limited

```
    P06     =       0;
    delay();
    P05     =       0;
    delay();
    P04     =       0;
    delay();
    P03     =       0;
    delay();
    P02     =       0;
    delay();
    P01     =       0;
    delay();
    P00     =       0;
    delay();

    P0      =       0xff;

    P40     =       0;
    delay();
    P41     =       0;
    delay();
    P42     =       0;
    delay();
    P43     =       0;
    delay();
    P44     =       0;
    delay();
    P45     =       0;
    delay();
    P46     =       0;
    delay();
    P47     =       0;
    delay();

    P4      =       0xff;

    P50     =       0;
    delay();
    P51     =       0;
    delay();
    P52     =       0;
    delay();
    P53     =       0;
    delay();
    P54     =       0;
    delay();
    P55     =       0;
    delay();

    P5      =       0xff;
```

STC MCU Limited

```
	while(1)
	{
		P1	=	0x00;
		delay();
		P1	=	0xff;

		P3	=	0x00;
		delay();
		P3	=	0xff;

		P2	=	0x00;
		delay();
		P2	=	0xff;

		P0	=	0x00;
		delay();
		P0	=	0xff;

		P4	=	0x00;
		delay();
		P4	=	0xff;

		P5	=	0x00;
		delay();
		P5	=	0xff;
	}
}

void delay(void)
{
	unsigned int i = 0;
	for(i=60000;i>0;i--)
	{
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
		_nop_();
```

STC MCU Limited

```
                _nop_();
                _nop_();
                _nop_();
                _nop_();

                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
        }
}
```

STC MCU Limited

4.9 I/O口各种不同的工作模式结构框图

4.9.1 准双向口输出配置

准双向口输出类型可用作输出和输入功能而不需重新配置口线输出状态。这是因为当口线输出为1时驱动能力很弱，允许外部装置将其拉低。当引脚输出为低时，它的驱动能力很强，可吸收相当大的电流。准双向口有3个上拉晶体管适应不同的需要。

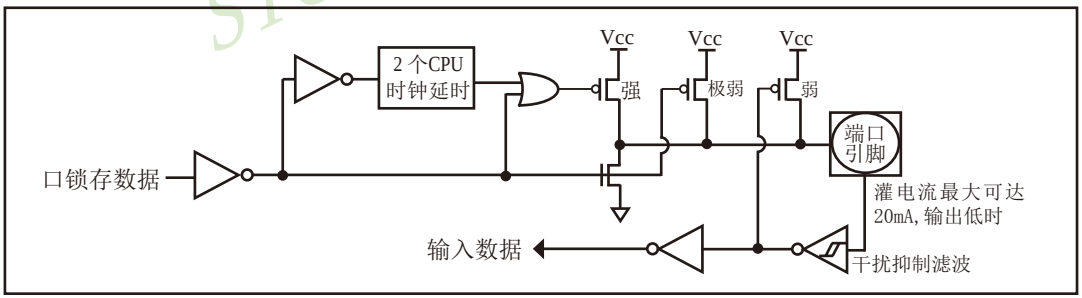
在3个上拉晶体管中，有1个上拉晶体管称为“弱上拉”，当口线寄存器为1且引脚本身也为1时打开。此上拉提供基本驱动电流使准双向口输出为1。如果一个引脚输出为1而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。**对于5V单片机，“弱上拉”晶体管的电流约250uA**；对于3.3V单片机，“弱上拉”晶体管的电流约150uA。

第2个上拉晶体管，称为“极弱上拉”，当口线锁存为1时打开。当引脚悬空时，这个极弱的上拉源产生很弱的上拉电流将引脚上拉为高电平。**对于5V单片机，“极弱上拉”晶体管的电流约18uA**；对于3.3V单片机，“极弱上拉”晶体管的电流约5uA。

第3个上拉晶体管称为“强上拉”。当口线锁存器由0到1跳变时，这个上拉用来加快准双向口由逻辑0到逻辑1转换。当发生这种情况时，强上拉打开约2个时钟以使引脚能够迅速地上拉到高电平。

准双向口输出如下图所示。

STC MCU Limited



准双口输出

STC 1T系列单片机为3.3V器件，如果用户在引脚加上5V电压，将会有电流从引脚流向 Vcc，这样导致额外的功率消耗。因此，建议不要在准双向口模式中向3.3V单片机引脚施加5V电压,如使用的话，要加限流电阻，或用二极管做输入隔离，或用三极管做输出隔离。

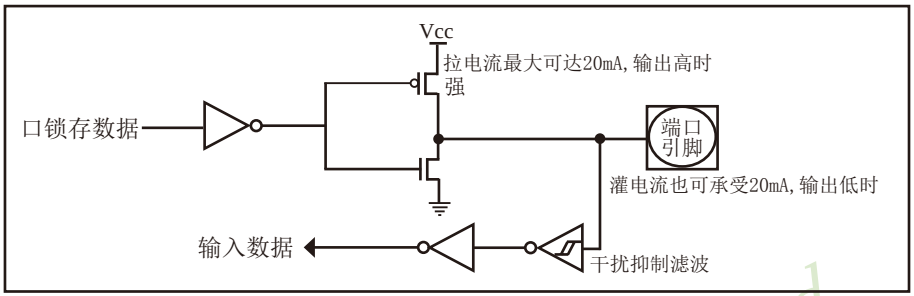
准双向口带有一个施密特触发输入以及一个干扰抑制电路。

准双向口读外部状态前,要先锁存为 ‘1’,才可读到外部正确的状态.

4.9.2 强推挽输出配置

强推挽输出配置的下拉结构与开漏输出以及准双向口的下拉结构相同，但当锁存器为1时提供持续的强上拉。推挽模式一般用于需要更大驱动电流的情况。

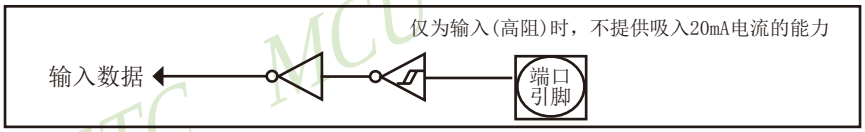
强推挽引脚配置如下图所示。



强推挽输出

4.9.3 仅为输入（高阻）配置

输入口配置如下图所示。



仅为输入(高阻)模式

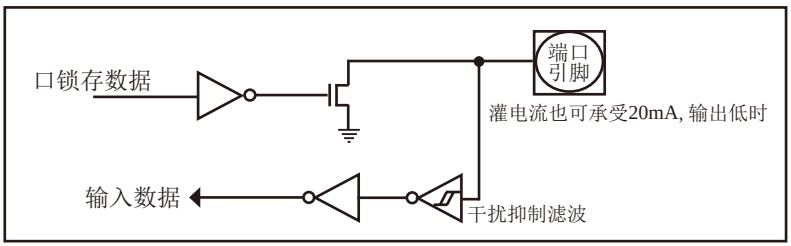
输入口带有一个施密特触发输入以及一个干扰抑制电路。

4.9.4 开漏输出配置(若外加上拉电阻，也可读外部状态或输出高电平)

开漏模式既可读外部状态也可对外输出(高电平或低电平)。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻。

当口线锁存器为0时，开漏输出关闭所有上拉晶体管。当作为一个逻辑输出高电平时，这种配置方式必须有外部上拉，一般通过电阻外接到Vcc。如果外部有上拉电阻，开漏的I/O口还可读外部状态，即此时被配置为开漏模式的I/O口还可作为输入I/O口。这种方式的下拉与准双向口相同。输出口线配置如下图所示。

开漏端口带有一个施密特触发输入以及一个干扰抑制电路。



开漏输出(如外部有上拉电阻，也可读外部状态或对外输出高电平)

STC MCU Limited

关于I/O口应用注意事项:

少数用户反映I/O口有损坏现象,后发现是

有些是I/O口由低变高读外部状态时,读不对,实际没有损坏,软件处理一下即可。

因为1T的8051单片机速度太快了,软件执行由低变高指令后立即读外部状态,此时由于实际输出还没有变高,就有可能读不对,正确的方法是在软件设置由低变高后加1到2个空操作指令延时,再读就对了.

有些实际没有损坏,加上拉电阻就OK了

有些是外围接的是NPN三极管,没有加上拉电阻,其实基极串多大电阻,I/O口就应该上拉多大的电阻,或者将该I/O口设置为强推挽输出.

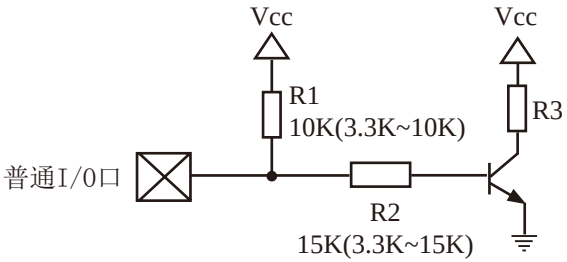
有些确实是损坏了,原因:

发现有些是驱动LED发光二极管没有加限流电阻,建议加1K以上的限流电阻,至少也要加470欧姆以上

发现有些是做行列矩阵按键扫描电路时,实际工作时没有加限流电阻,实际工作时可能出现2个I/O口均输出为低,并且在按键按下时,短接在一起,我们知道一个CMOS电路的2个输出脚不应该直接短接在一起,按键扫描电路中,此时一个口为了读另外一个口的状态,必须先置高才能读另外一个口的状态,而8051单片机的弱上拉口在由0变为1时,会有2个时钟的强推挽高输出电流,输出到另外一个输出为低的I/O口,就有可能造成I/O口损坏.建议在两侧各加300欧姆限流电阻,或者在软件处理上,不要出现按键两端的I/O口同时为低.

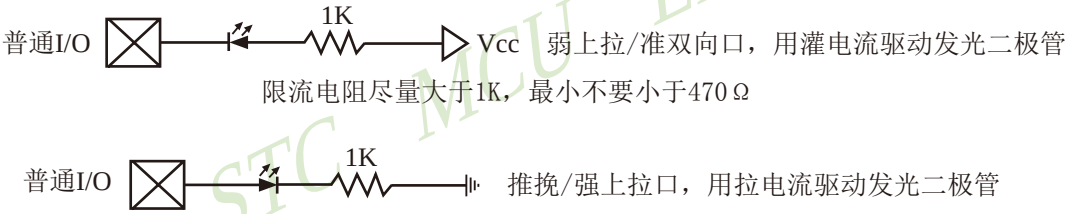
STC MCU Limited

4.10 一种典型三极管控制电路



如果用弱上拉控制，建议加上拉电阻R1(3.3K～10K)，如果不加上拉电阻R1(3.3K～10K)，建议R2的值在15K以上，或用强推挽输出。

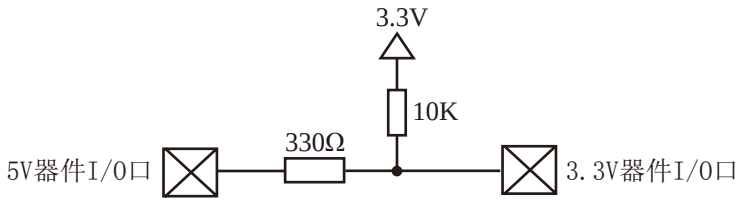
4.11 典型发光二极管控制电路



STC MCU Limited

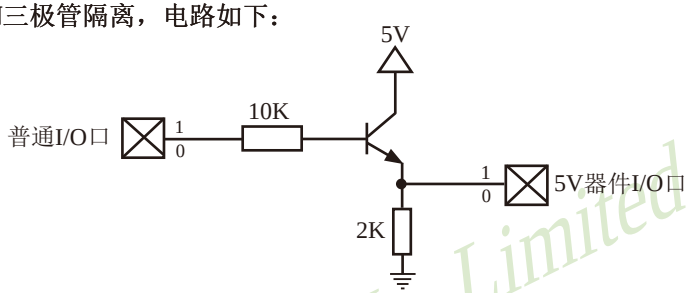
4.12 混合电压供电系统3V/5V器件I/O口互连

STC 1T系列5V单片机连接3.3V器件时，为防止3.3V器件承受不了5V，可将相应的5V单片机I/O口先串一个330Ω的限流电阻到3.3V器件I/O口，程序初始化时将5V器件的I/O口设置成开漏配置，断开内部上拉电阻，相应的3.3V器件I/O口外部加10K上拉电阻到3.3V器件的Vcc，这样高电平是3.3V，低电平是0V，输入输出一切正常。



STC 1T系列3V单片机连接5V器件时，为防止3V器件承受不了5V，如果相应的I/O口是输入，可在该I/O口上串接一个隔离二极管，隔离高压部分。外部信号电压高于单片机工作电压时截止，I/O口因内部上拉到高电平，所以读I/O口状态是高电平；外部信号电压为低时导通，I/O口被钳位在0.7V，小于0.8V时单片机读I/O口状态是低电平。

STC 1T系列3V单片机连接5V器件时，为防止3V器件承受不了5V，如果相应的I/O口是输出，可用一个NPN三极管隔离，电路如下：



STC MCU Limited

4.13 I/O口的外部输入何时低(0.8V以下)何时高电平(2.2V以上)

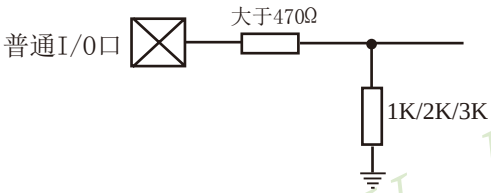
当I/O口的外部输入电平在0.8V以下时，则单片机认为该I/O口的外部输入为低电平；当I/O口的外部输入电平在2.2V以上时，则单片机认为该I/O口的外部输入为高电平。

实际制造时按I/O口的外部输入电平在1.2V以下时为低电平，在1.8V以上时为高电平。但由于存在制造误差，1.2V以下单片机不一定认为I/O口的外部输入为低电平，1.8V以上单片机也不一定就认为I/O口的外部输入为高电平。但我们保证0.8V以下可以为低电平，2.2V以上可以为高电平，外部输入电平在0.8V~2.2V之间不保证单片机能固定地识别I/O口的外部输入为低电平还是为高电平。

4.14 如何让I/O口上电复位时为低电平

普通8051单片机上电复位时普通I/O口为弱上拉高电平输出,而很多实际应用要求上电时某些I/O口为低电平输出,否则所控制的系统(如马达)就会误动作,现STC15系列单片机由于既有弱上拉输出又有强推挽输出,就可以很轻松的解决此问题。

现可在STC15系列单片机I/O口上加一个下拉电阻(1K/2K/3K),这样上电复位时，虽然单片机内部I/O口是弱上拉/高电平输出，但由于内部上拉能力有限，而外部下拉电阻又较小,无法将其拉高,所以该I/O口上电复位时外部为低电平。如果要将此I/O口驱动为高电平，可将此I/O口设置为强推挽输出，而强推挽输出时，I/O口驱动电流可达20mA,故肯定可以将该口驱动为高电平输出。



特别提示：STC15系列单片机的RSTOUT_LOW管脚可以在ISP烧录程序时设置上电复位后输出低电平还是高电平，其他管脚上电复位后均输出高电平。

STC15系列有的单片机(如STC15W4K60S4及STC15F2K60S2系列)的RSTOUT_LOW脚在P2.0口(P2.0/RSTOUT_LOW)，有的单片机(如STC15W201S系列)的RSTOUT_LOW脚在P1.0口(P1.0/RSTOUT_LOW)，有的单片机(如STC15F101W系列)的RSTOUT_LOW脚在P3.3口(P3.3/RSTOUT_LOW).

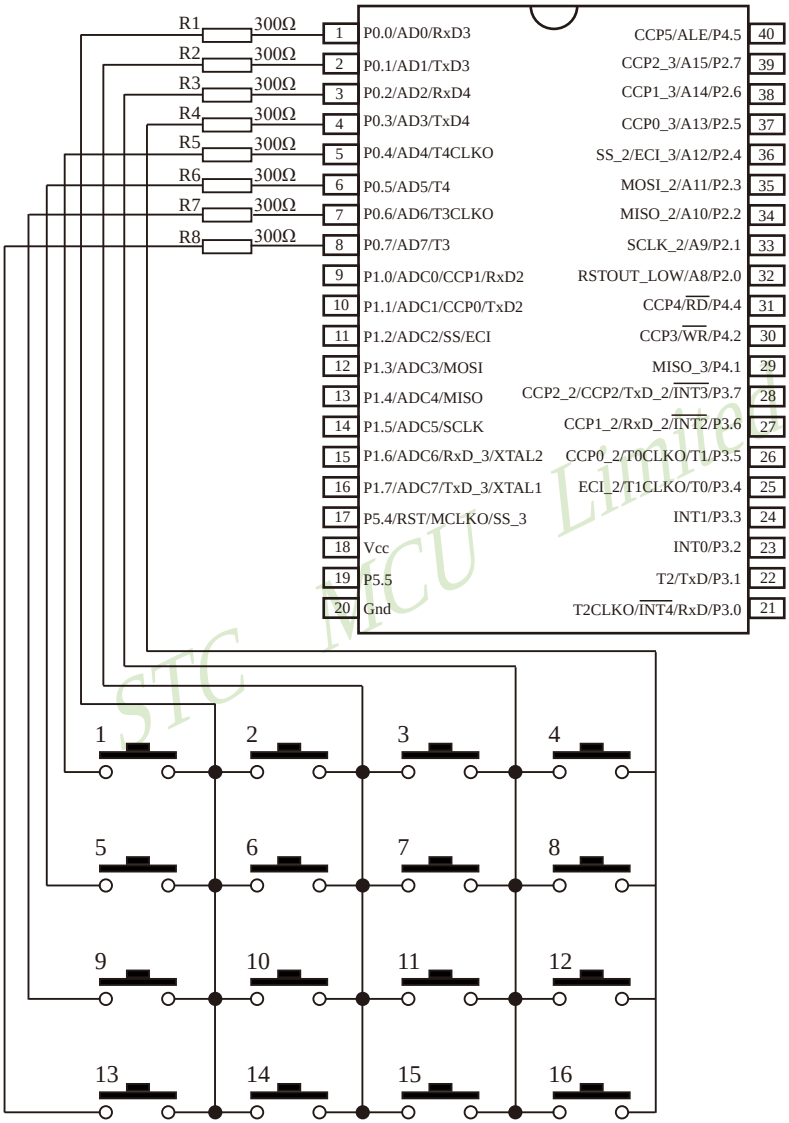
STC MCU Limited

4.15 PWM输出时I/O口的状态

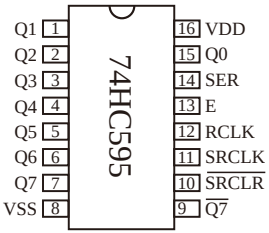
当I/O口作为PWM输出用时，不改变口的输出状态，要软件设置，建议用户将作PWM用的I/O口状态设置成强推挽输出，与早期的STC 1T系列单片机(如STC12系列)不同。下表为STC12系列单片机的I/O口作PWM用时，该口的状态：

PWM 之前口的状态	PWM时口的状态
弱上拉/准双向口	强推挽输出/强上拉输出 要加输出限流电阻10K ~ 1K
强推挽输出	强推挽输出/强上拉输出 要加输出限流电阻10K ~ 1K
仅为输入/高阻	PWM无效
开漏	开漏

4.16 I/O口行列式按键扫描应用线路图



4.17 74HC595管脚介绍及逻辑表



74HC595的管脚图

74HC595管脚介绍		
管脚名称	管脚编号	管脚功能
Q0 ~ Q7	15, 1~7	并行数据输出
$\overline{Q7}$	9	串行数据输出
SRCLR	10	清除端(低电平有效)
SRCLK	11	移位寄存器的时钟输入
RCLK	12	三态输出锁存器的时钟输入
E	13	输出允许控制
SER	14	串行数据输入
VDD	16	电源正极
VSS	8	电源接地端

单片机系统中，常采用74HC595作为LED的静态显示接口。74HC595的管脚如上图所示。该芯片内含8位串入、串/并出移位寄存器和8位三态输出锁存器。寄存器和锁存器分别有各自的时钟输入(SRCLK和RCLK)，都是上升沿有效。当SRCLK从低到高电平跳变时，串行输入数据(SER)移入寄存器；当RCLK从低到高电平跳变时，寄存器的数据置入锁存器。清除端(SRCLR)的低电平只对寄存器复位($\overline{Q7}$为低电平)，而对锁存器无影响。当输出允许控制(E)为高电平时，并行输出(Q0~Q7)为高阻态，而串行输出($\overline{Q7}$)不受影响。

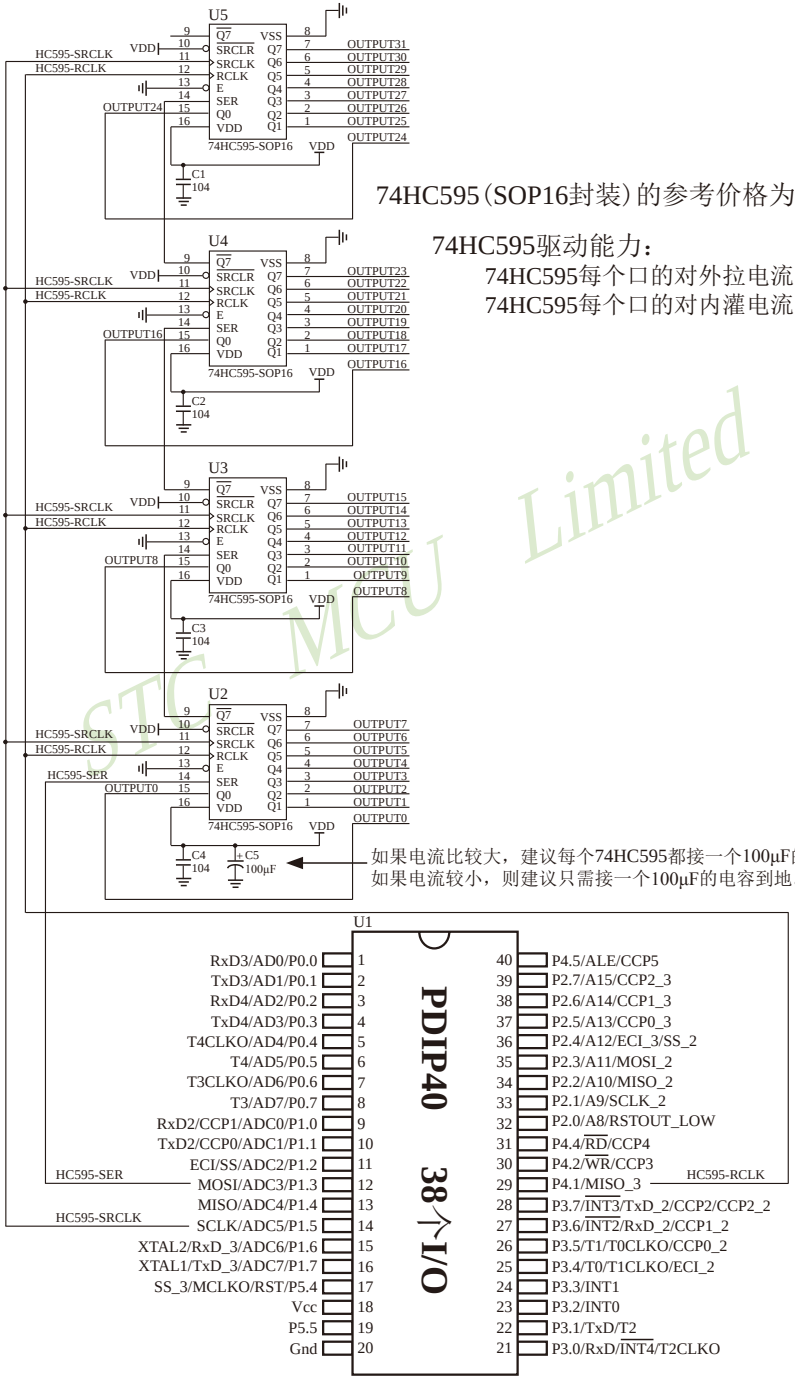
74HC595最多需要5根控制线，即SER、SRCLK、RCLK、SRCLR和E。其中SRCLR可以直接接到高电平，用软件来实现寄存器清零；如果不需要软件改变亮度，E可以直接接到低电平，而用硬件来改变亮度。把其余三根线和单片机的I/O口相接，即可实现对LED的控制。

数据从SDI口送入74HC595，在每个SRCLK的上升沿，SER口上的数据移入寄存器，在SRCLK的第9个上升沿，数据开始从$\overline{Q7}$移出。如果把第一个74HC595的$\overline{Q7}$和第二个74HC595的SER相接，数据即移入第二个74HC595中，照此一个一个接下去，可任意多个。数据全部传送完后，给RCLK一个上升沿，寄存器中的数据即置入锁存器中。此时如果E为低电平，数据即从并口Q0~Q7输出，把Q0~Q7与LED的8段相接，LED就可以显示了。想要软件改变LED亮度，只需改变E的占空比就行了。

74HC595逻辑表					
输入管脚					输出管脚
SER	SRCLK	SRCLR	RCLK	E	
X	X	X	X	H	Q0~Q7输出高阻
X	X	X	X	L	Q0~Q7输出有效值
X	X	L	X	X	移位寄存器清零
L	上升沿	H	X	X	移位寄存器存储L
H	上升沿	H	X	X	移位寄存器存储H
X	下降沿	H	X	X	移位寄存器状态保持
X	X	X	上升沿	X	输出存储器锁存移位寄存器中的状态值
X	X	X	下降沿	X	输出存储器状态保持

STC MCU Limited

4.18 利用74HC595扩展I/O口的线路图(串行扩展，3根线)



74HC595(SOP16封装)的参考价格为RMB 0.2元。

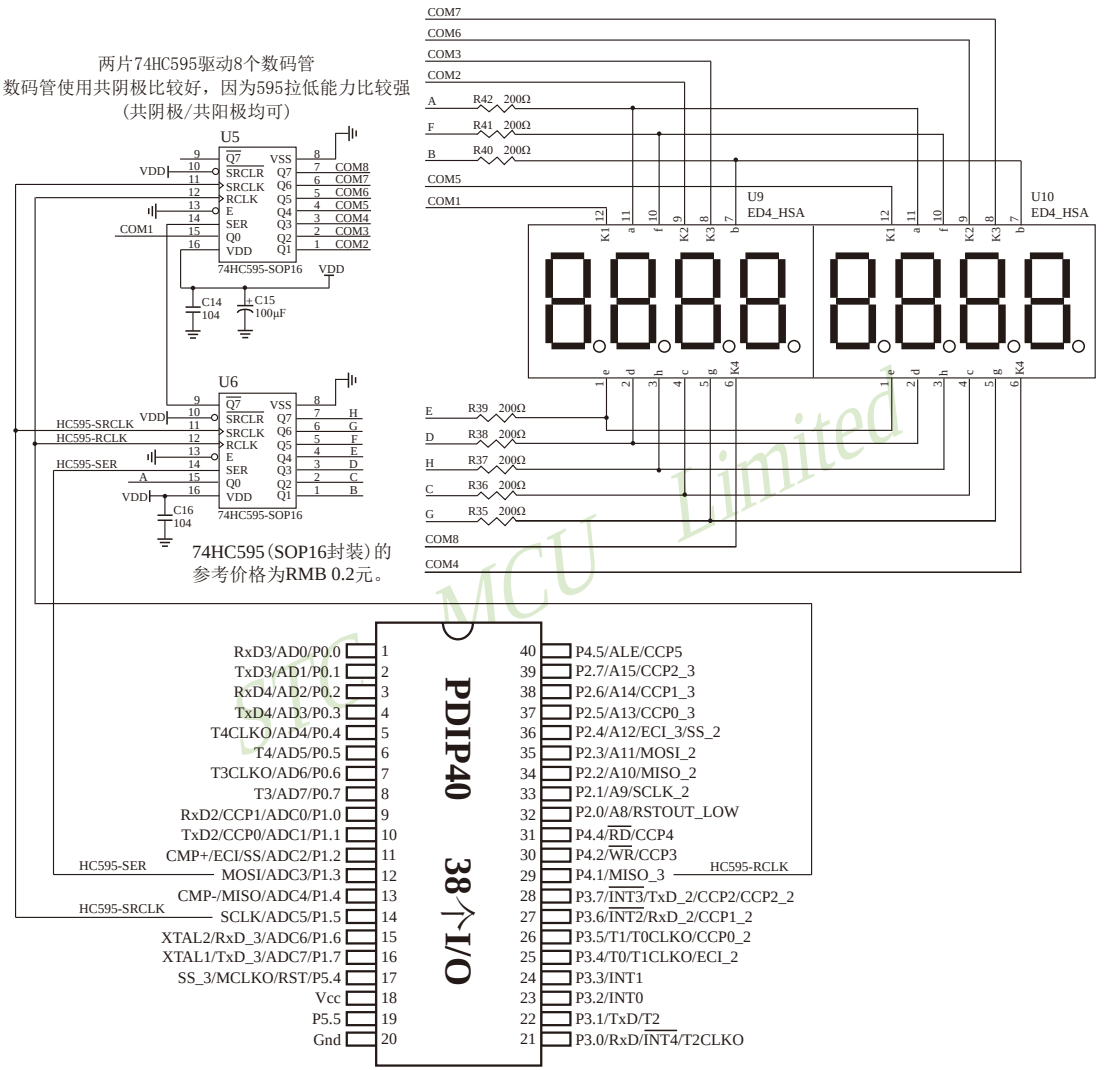
74HC595驱动能力：

74HC595每个口的对外拉电流能力为30mA；

74HC595每个口的对内灌电流能力为100mA。

STC MCU Limited

4.19 利用74HC595驱动8个数码管(串行扩展,3根线)的线路图



STC MCU Limited

4.20 利用普通I/O口控制74HC595驱动8个数码管的测试程序

1. C程序

```
/*---------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------*/
/* --- STC 1T Series MCU Programme Demo ------------------------------*/
/* --- Fax: 86-755-82944243 ------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------*/

/* 本程序经过测试完全正常, 不提供电话技术支持, 如不能理解, 请自行补充相关基础.  */

/*************    本程序功能说明    **************

用STC的MCU的普通IO方式控制74HC595驱动8位数码管。

用户可以修改宏来选择时钟频率.

用户可以在显示函数里修改成共阴或共阳.推荐尽量使用共阴数码管.

显示效果为: 8个数码管循环显示0,1,2...,A,B..F,消隐.

*****************************************/

#include  "reg52.h"

/****************************** 用户定义宏 ***********************************/

#define    MAIN_Fosc        11059200UL                //定义主时钟
//#define  MAIN_Fosc        22118400UL                //定义时钟

/*****************************************************************************/


/********************** 下面的宏自动生成, 用户不可修改 ***************************/

#define    Timer0_Reload      (MAIN_Fosc / 12000)

/*****************************************************************************/

/*************    本地常量声明      **************/
unsigned char code t_display[]={
//          0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F  消隐
        0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,0x00};
                                                                                //段码
```

STC MCU Limited

```
unsigned char code T_COM[]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};          //位码

/**************    本地变量声明    ***************/
//sbit    P_HC595_SER   =   P3^2;               //pin 14   SER     data input
//sbit    P_HC595_RCLK  =   P3^4;               //pin 12   RCLk    store (latch) clock
//sbit    P_HC595_SRCLK =   P3^3;               //pin 11   SRCLK   Shift data clock

sbit    P_HC595_SER   =   P1^3;               //pin 14   SER     data input
sbit    P_HC595_RCLK  =   P4^1;               //pin 12   RCLk    store (latch) clock
sbit    P_HC595_SRCLK =   P1^5;               //pin 11   SRCLK   Shift data clock

unsigned char   LED8[8];                      //显示缓冲
unsigned char   display_index;                //显示位索引
bit     B_1ms;                                //1ms标志

/*********************************************/
void main(void)
{
        unsigned char   i,  k;
        unsigned int    j;

        TMOD    =   0x01;                     //Timer 0 config as 16bit timer, 12T
        TH0     =   (65536 - Timer0_Reload) / 256;
        TL0     =   (65536 - Timer0_Reload) % 256;
        ET0     =   1;
        TR0     =   1;
        EA      =   1;

        for(i=0; i<8; i++)  LED8[i] = 0x10;   //上电消隐
        j = 0;
        k = 0;
//      for(i=0; i<8; i++)  LED8[i] = i;

        while(1)
        {
                while(!B_1ms);                //等待1ms到
                B_1ms = 0;
                if(++j >= 500)                //500ms到
                {
                        j = 0;
                        for(i=0; i<8; i++)  LED8[i] = k;     //刷新显示
                        if(++k > 0x10)      k = 0;          //8个数码管循环显示0,1,2...,A,B..F,消隐.
                }
        }
}

/*********************************************/
```

STC MCU Limited

```
/*********************************************/
void Send_595(unsigned char dat)        //发送一个字节
{
        unsigned char        i;
        for(i=0; i<8; i++)
        {
                if(dat & 0x80)       P_HC595_SER = 1;
                else                 P_HC595_SER = 0;
                P_HC595_SRCLK = 1;
                P_HC595_SRCLK = 0;
                dat = dat << 1;
        }
}

/*********************************************/
void DisplayScan(void)        //显示扫描函数
{
//      Send_595(~T_COM[display_index]);                    //共阴    输出位码
//      Send_595(t_display[LED8[display_index]]);           //共阴    输出段码
        Send_595(T_COM[display_index]);                     //共阳    输出位码
        Send_595(~t_display[LED8[display_index]]);          //共阳    输出段码
        P_HC595_RCLK = 1;
        P_HC595_RCLK = 0;                                   //锁存输出数据
        if(++display_index >= 8)      display_index = 0;    //8位结束回0
}


/*********************************************/
void timer0 (void) interrupt 1 //Timer0 1ms中断函数
{
        TH0  = (65536 - Timer0_Reload) / 256;               //重装定时值
        TL0  = (65536 - Timer0_Reload) % 256;

        DisplayScan();                                      //1ms扫描显示一位
        B_1ms = 1;                                          //1ms标志
}
```

STC MCU Limited

2. 汇编程序

```
;/*---------------------------------------------------------------------*/
;/* --- STC MCU International Limited ---------------------------------*/
;/* --- STC 1T Series MCU Programme Demo ------------------------------*/
;/* --- Fax: 86-755-82944243 ------------------------------------------*/
;/* --- Tel: 86-755-82948412 ------------------------------------------*/
;/* --- Web: www.STCMCU.com -------------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------------*/

;/*    本程序经过测试完全正常, 不提供电话技术支持, 如不能理解, 请自行补充相关基础.  */

;/**************  本程序功能说明  ***************

;用STC的MCU的任意IO方式控制74HC595驱动8位数码管。

;用户可以在显示函数里修改成共阴或共阳.推荐尽量使用共阴数码管.

;显示效果为: 8个数码管循环显示0,1,2...,A,B..F,消隐.

;*****************************************/
;定义Timer0 1ms重装值
D_Timer0_Reload         EQU     (0-921)                 ;1ms for 11.0592MHZ
//D_Timer0_Reload       EQU     (0-1832)                ;1ms for 22.1184MHZ

;/**************  本地变量声明  **************/
;P_HC595_SER   BIT       P3.2                            ;pin 14    SER     data input
;P_HC595_RCLK  BIT       P3.4                            ;pin 12    RCLk    store (latch) clock
;P_HC595_SRCLK BIT       P3.3                            ;pin 11    SRCLK   Shift data clock

P_HC595_SER   BIT       P1.3                            ;pin 14    SER     data input
P_HC595_RCLK  BIT       P4.1                            ;pin 12    RCLk    store (latch) clock
P_HC595_SRCLK BIT       P1.5                            ;pin 11    SRCLK   Shift data clock

LED8            EQU     030H
display_index   DATA    038H
FLAG0           DATA    20H
B_1ms           BIT     FLAG0.0

;**********************************************************************
;**********************************************************************
        ORG     00H                                     ;reset
        LJMP    F_MAIN_FUNC

        ORG     03H                                     ;INT0 interrupt
;       LJMP    F_INT0_interrupt
        RETI
```

STC MCU Limited

```
        ORG     0BH                             ;Timer0 interrupt
        LJMP    F_Timer0_interrupt
        RETI

        ORG     13H                             ;INT1 interrupt
;       LJMP    F_INT1_interrupt

        ORG     1BH                             ;Timer1 interrupt
;       LJMP    F_Timer1_interrupt
        RETI

;**********************************************************************
;**********************************************************************

;/*********************************************/
F_MAIN_FUNC:
        MOV     SP,     #50H

        MOV     TMOD,   #01H                    ;Timer 0 config as 16bit timer, 12T
        MOV     TH0,    #HIGH D_Timer0_Reload   ;1ms
        MOV     TL0,    #LOW  D_Timer0_Reload
        SETB    ET0                             ;
        SETB    TR0                             ;
        SETB    EA                              ;

        MOV     R0,     #LED8
L_InitLoop1:
        MOV     @R0,    #10H                    ;上电消隐
        INC     R0
        MOV     A,R0
        CJNE    A,      #(LED8+8),      L_InitLoop1

        MOV     R2,     #HIGH   500             ;500ms
        MOV     R3,     #LOW    500
        MOV     R4,     #0
L_MainLoop:
        JNB     B_1ms,  $                       ;等待1ms到
        CLR     B_1ms

        MOV     A,      R3
        CLR     C
        SUBB    A,      #1
        MOV     R3,     A
        MOV     A,      R2
        SUBB    A,      #0
        MOV     R2,     A
        ORL     A,      R3
        JNZ     L_MainLoop
```

STC MCU Limited

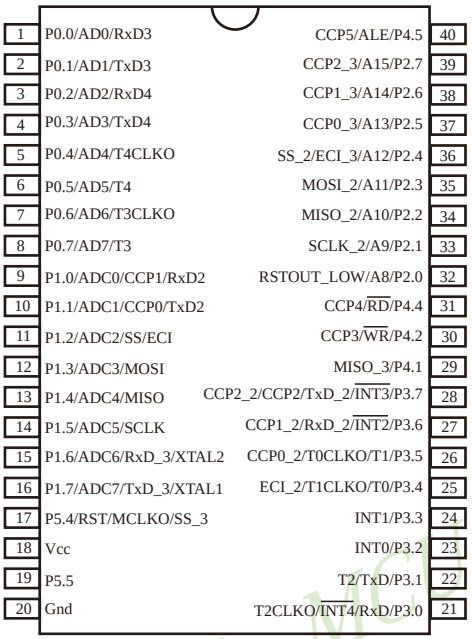
```
        MOV     R2,     #HIGH   500                     ;500ms
        MOV     R3,     #LOW    500

        MOV     R0,     #LED8                           ;刷新显示
L_OptionLoop1:
        MOV     A,      R4
        MOV     @R0,    A                               ;
        INC     R0
        MOV     A,      R0
        CJNE    A,      #(LED8+8),      L_OptionLoop1
        INC     R4      ;
        MOV     A,      R4
        CJNE    A,      #11H,   L_MainLoop              ;8个数码管循环显示0,1,2...,A,B..F,消隐.
        MOV     R4,     #0
        SJMP    L_MainLoop

;/*********************************************/


t_display:
; 0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F   消隐
DB 03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H,000H
;段码

T_COM:
        DB      01H,02H,04H,08H,10H,20H,40H,80H         ;位码


;/*********************************************/
F_Send_595:                                             ;发送一个字节
        MOV     R0,     #8
L_Send595_Loop:
        RLC     A
        MOV     P_HC595_SER,C
        SETB    P_HC595_SRCLK
        CLR     P_HC595_SRCLK
        DJNZ    R0,     L_Send595_Loop
        RET

;/*********************************************/
F_DisplayScan:                                          ;显示扫描函数
        MOV     DPTR,   #T_COM
        MOV     A,      display_index
        MOVC    A,      @A+DPTR
;       CPL             A                               ;共阴      共阳时注释掉本句
        LCALL   F_Send_595                              ;输出位码

        MOV     DPTR,   #t_display
```

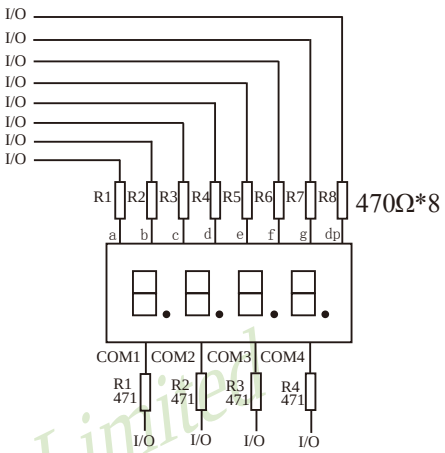
STC MCU Limited

```
        MOV     A,      #LED8
        ADD     A,      display_index
        MOV     R0,     A
        MOV     A,      @R0
        MOVC    A,      @A+DPTR
        CPL     A                               ;共阳     共阴时注释掉本句
        LCALL   F_Send_595                      ;输出段码
        SETB    P_HC595_RCLK
        CLR     P_HC595_RCLK    ;锁存输出数据
        INC     display_index
        MOV     A,      display_index
        CJNE    A,      #8,L_QuitDisplayScan
        MOV     display_index,  #0              ;8位结束回0
L_QuitDisplayScan:
        RET

;**********************************************************************
;**********************************************************************
F_Timer0_interrupt:                             ;Timer0 1ms中断函数
        PUSH    PSW                             ;现场保护
        PUSH    ACC
        MOV     A,      R0
        PUSH    ACC
        PUSH    DPH
        PUSH    DPL

        MOV     TH0,    #HIGH D_Timer0_Reload   ;1ms      重装定时值
        MOV     TL0,    #LOW  D_Timer0_Reload

        LCALL   F_DisplayScan                   ;1ms扫描显示一位
        SETB    B_1ms                           ;1ms标志

L_QuitT0Interrupt:
        POP     DPL                             ;现场恢复
        POP     DPH
        POP     ACC
        MOV     R0,A
        POP     ACC
        POP     PSW
        RETI


        END
```

STC MCU Limited

4.21 I/O口直接驱动LED数码管应用线路图

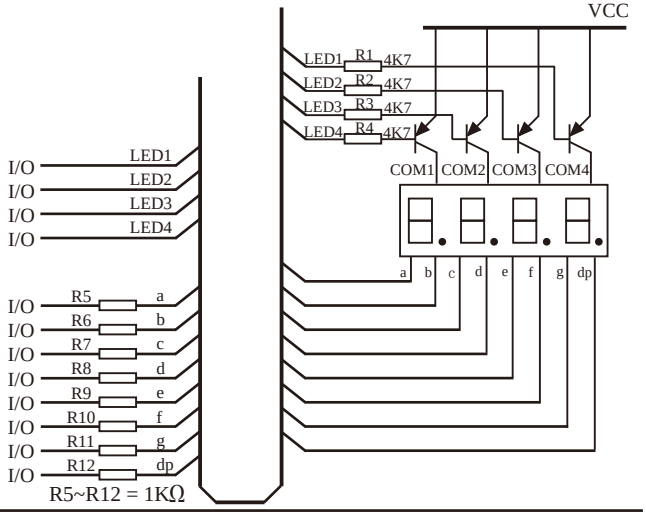




I/O口动态扫描驱动4个共阴极数码管参考电路图

I/O 口动态扫描驱动数码时，可以一次点亮一个数码管中的8段，但为降低功耗，建议可以一次只点亮其中的4段或者2段

I/O 口动态扫描驱动4个共阳极数码管参考电路图

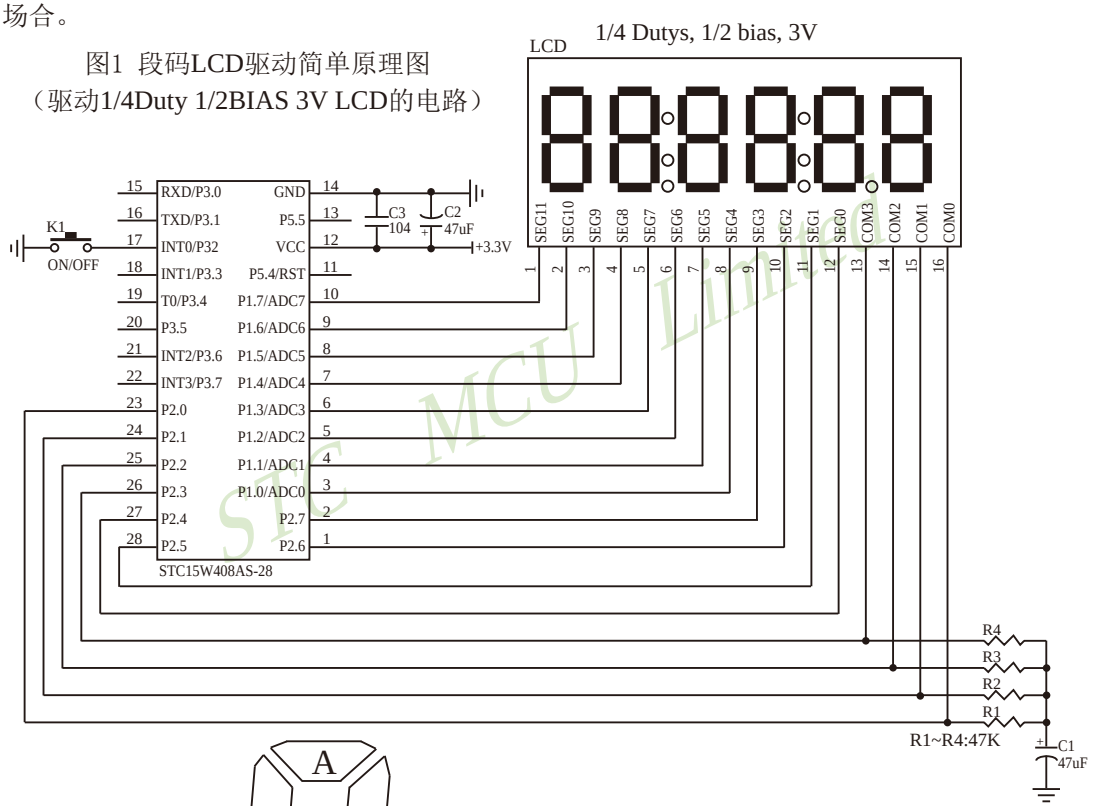


STC MCU Limited

4.22 用STC MCU的I/O口直接驱动段码LCD的原理及扫描程序

当产品需要段码LCD显示时，如果使用不带LCD驱动器的MCU，则需要外接LCD驱动IC，这会增加成本和PCB面积。事实上，很多小项目，比如大量的小家电，需要显示的段码不多，常见的是4个8带小数点或时钟的冒号“：”，这样如果使用IO口直接扫描显示，则会减小PCB面积，降低成本。因此提出使用STC MCU I/O口直接驱动段码LCD的方案，段码LCD驱动简单原理如下图1。但是，本方案不合适驱动太多的段（占用IO太多），也不合适非常低功耗的场合。

图1 段码LCD驱动简单原理图
（驱动1/4Duty 1/2BIAS 3V LCD的电路）



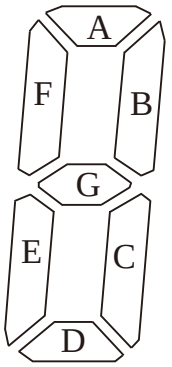
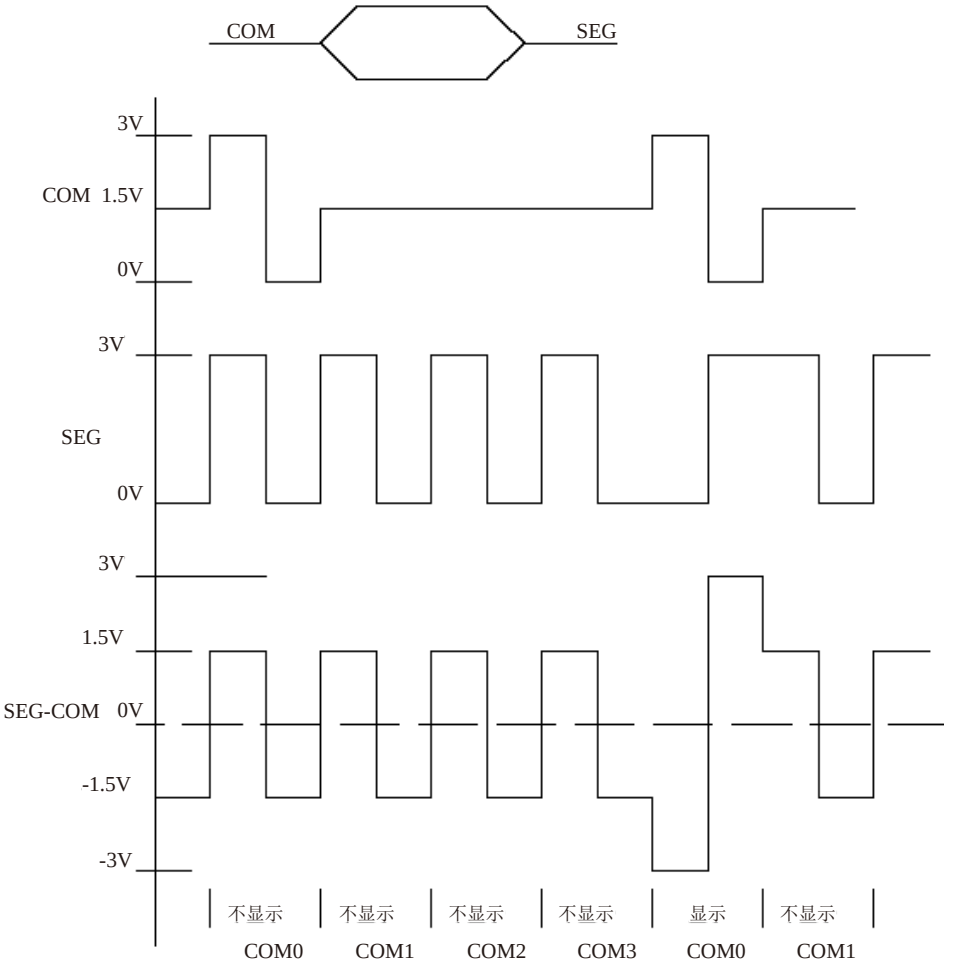


图2 段码名称图

LCD是一种特殊的液态晶体，在电场的作用下晶体的排列方向会发生扭转，因而改变其透光性，从而可以看到显示内容。LCD有一个扭转阀值，当LCD两端电压高于此阀值时，显示内容，低于此阀值时，不显示。通常LCD有3个参数：工作电压、DUTY（对应COM数）和BIAS（即偏压，对应阀值），比如4.5V、1/4 DUTY、1/3 BIAS，表示LCD显示电压为4.5V，4个COM，阀值大约是1.5V，当加在某段LCD两端电压大于1.5V时（一般加4.5V）显示，而加1.5V时不显示。但是LCD对于驱动电压的反应不是很明显的，比如加2V时，可能会微弱显示，这就是通常说的“鬼影”。所以要保证驱动显示时，要大于阀值电压比较多，而不显示时，要用比阀值小比较多的电压。

注意：LCD的两端不能加直流电压，否则时间稍长就会损坏，所以要保证加在LCD两端的驱动电压的平均电压为0。LCD使用时分割扫描法，任何时候一个COM扫描有效，另外的COM处于无效状态。

图3　1/4Duty 1/2BIAS LCD 扫描原理图



驱动1/4Duty 1/2BIAS 3V的方案电路见图1，LCD扫描原理见图3，MCU为3V工作，用双向口做COM，PUSH-PULL或STANDARD输出口接SEG，并且每个COM都接一个47K电阻到一个电容，RC滤波后得到一个中点电压。在轮到某个COM扫描时，设置成PUSH-PULL输出，如果与本COM连接的SEG不显示，则SEG输出与COM同相，如果显示，则反相。扫描完后，这个COM的IO就设置成高阻，这样这个COM就通过47K电阻连接到1/2VDD电压，而SEG继续输出方波，这样加在LCD上的电压，显示时是+-VDD，不显示时是+-1/2VDD，保证了LCD两端平均直流电压为0。

驱动1/4Duty 1/3BIAS 3V的方案电路见图4，LCD扫描原理见图5，MCU为5V工作，SEG线通过电阻分压输出1.5V、3.5V，COM线通过电阻分压输出0.5V、2.5V（高阻时）、4.5V。在轮到某个COM扫描时，设置成PUSH-PULL输出，如果与本COM连接的SEG不显示，则SEG输出与COM同相，如果显示，则反相。扫描完后，这个COM的IO就设置成高阻，这样这个COM就通过47K电阻连接到2.5V电压，而SEG继续输出方波，这样加在LCD上的电压，显示时是+-3.0V，不显示时是+-1.0V，完全满足LCD的扫描要求。

当需要睡眠省电时，把所有COM和SEG驱动IO全部输出低电平，LCD驱动部分不会增加额外电流。

STC MCU Limited

图4 驱动1/4Duty 1/3BIAS 3V LCD的电路图

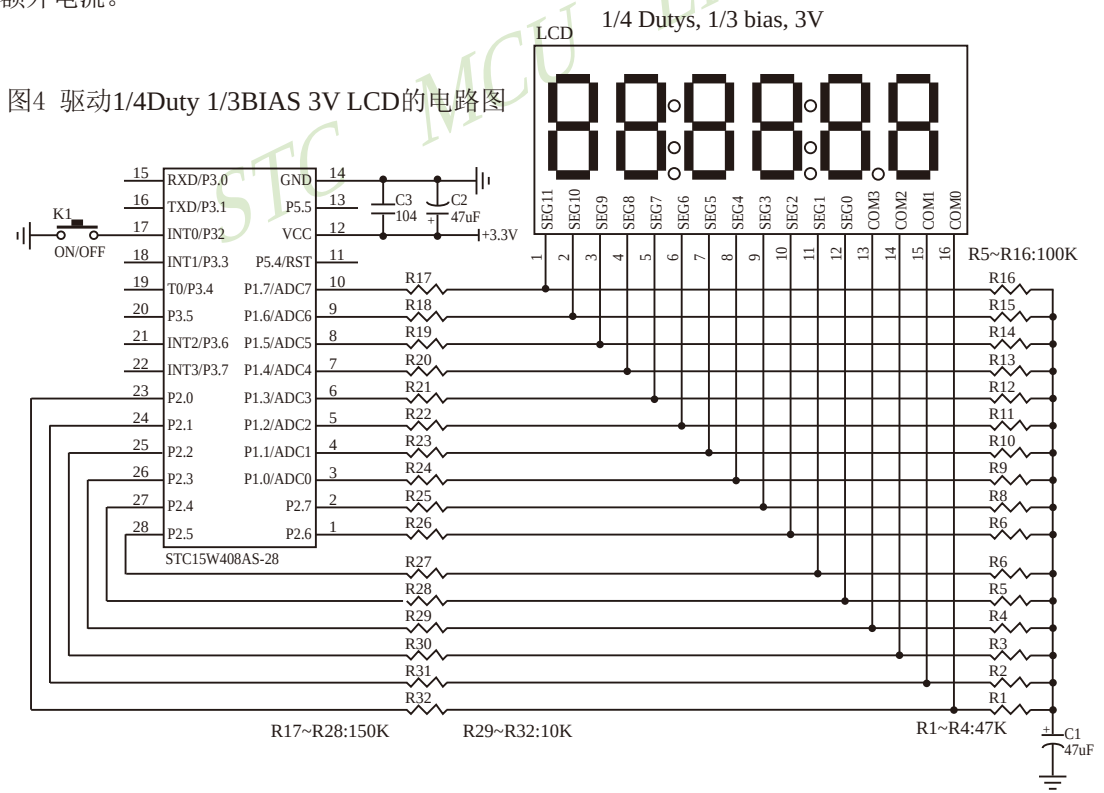
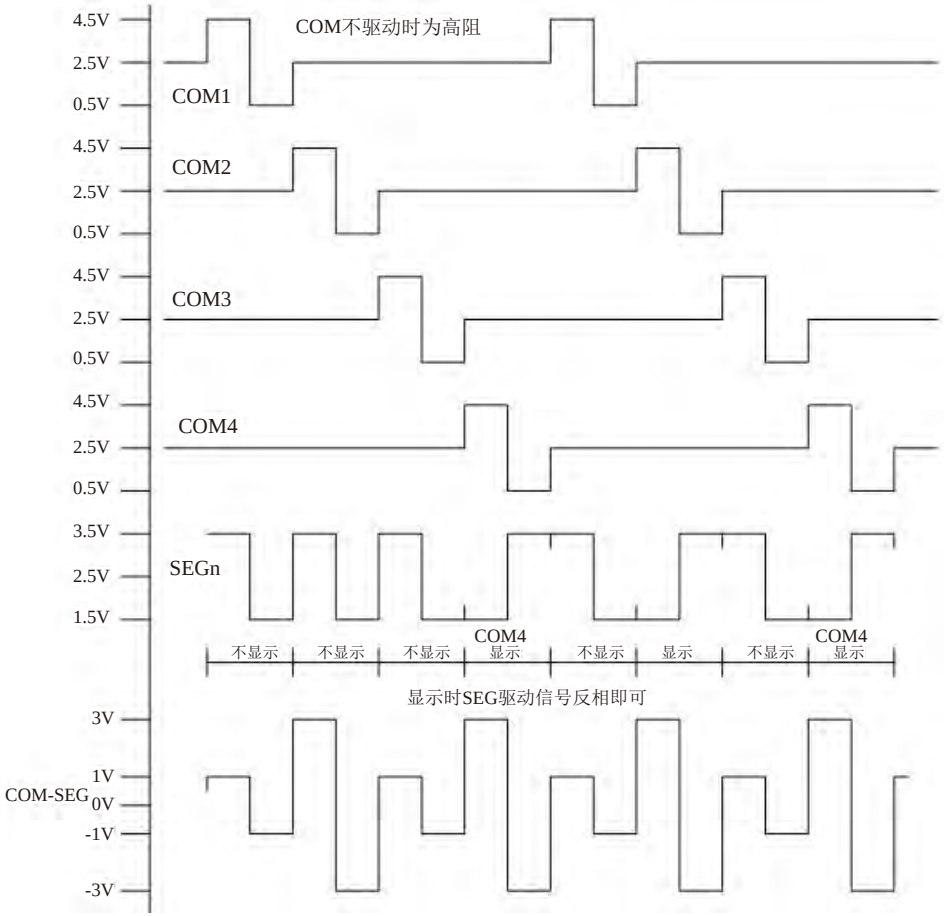


图5　1/4Duty 1/3BIAS LCD 扫描原理图



为了使用方便，显示内容放在一个显存中，其中的各个位与LCD的段一一对应，见图6。

图6：LCD真值表和显存影射表

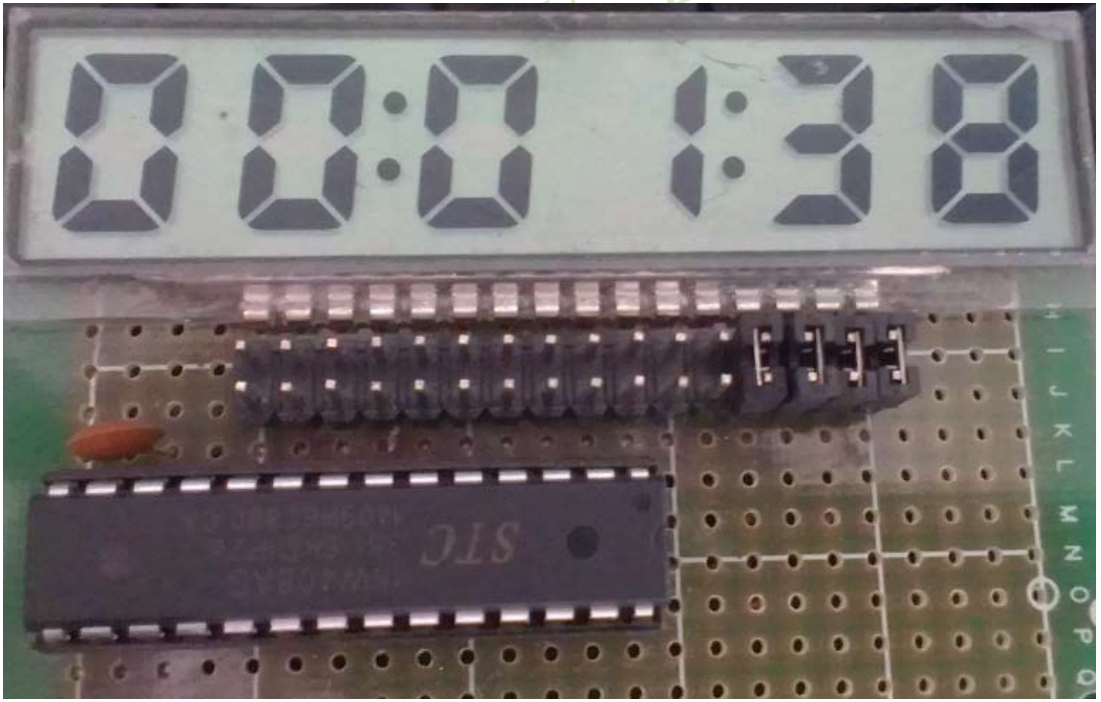
LCD真值表：

MCU PIN	P17	P16	P15	P14	P13	P12	P11	P10	P27	P26	P25	P24	P23	P22	P21	P20
LCD PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LCD PIN name	SEG11	SEG10	SEG9	SEG8	SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0	COM3	COM2	COM1	COM0
	--	1D	2:	2D	2.	3D	4:	4D	4.	5D	5.	6D	COM3			
	1E	1C	2E	2C	3E	3C	4E	4C	5E	5C	6E	6C		COM2		
	1G	1B	2G	2B	3G	3B	4G	4B	5G	5B	6G	6B			COM1	
	1F	1A	2F	2A	3F	3A	4F	4A	5F	5A	6F	6A				COM0

显存影射表：

	B7	B6	B5	B4	B3	B2	B1	B0
buff[0]:	--	1D	2:	2D	2.	3D	4:	4D
buff[1]:	1E	1C	2E	2C	3E	3C	4E	4C
buff[2]:	1G	1B	2G	2B	3G	3B	4G	4B
buff[3]:	1F	1A	2F	2A	3F	3A	4F	4A
buff[4]:	4.	5D	5.	6D	--	--	--	--
buff[5]:	5E	5C	6E	6C	--	--	--	--
buff[6]:	5G	5B	6G	6B	--	--	--	--
buff[7]:	5F	5A	6F	6A	--	--	--	--

图7：驱动效果照片



本LCD扫描程序仅需要两个函数：

1、 LCD段码扫描函数 void　　LCD_scan(void)

程序隔一定的时间调用这个函数，就会将LCD显示缓冲的内容显示到LCD上，全部扫描一次需要8个调用周期，调用间隔一般是1~2ms，假如使用1ms，则扫描周期就是8ms，刷新率就是125HZ。

2、 LCD段码显示缓冲装载函数 void　LCD_load(u8 n,u8 dat)

本函数用来将显示的数字或字符放在LCD显示缓冲中，比如LCD_load(1,6)，就是要在第一个数字位置显示数字6，支持显示0~9，A~F，其它字符用户可以自己添加。

另外，用宏来显示、熄灭或闪烁冒号或小数点。

```
/****************** LCD段码扫描函数 ***************************
u8      code T_COM[4]    =      {0x08,0x04,0x02,0x01};
void    LCD_scan(void)                                  //5us @22.1184MHZ
{
        u8      j;
        j = scan_index >> 1;                            //COMx
        P2n_pure_input(0x0f);                           //全部COM输出高阻, COM为中点电压

        if(scan_index & 1)                              //反相扫描
        {       P1 = ~LCD_buff[j];                      //送SEG驱动码
                P2 = ~(LCD_buff[j|4] & 0xf0);           //送SEG驱动码和COM驱动码
        }
        else                                            //正相扫描
        {       P1 = LCD_buff[j];                       //送SEG驱动码
                P2 = LCD_buff[j|4] & 0xf0;              //送SEG驱动码和COM驱动码
        }

        P2n_push_pull(T_COM[j]);                        //某个COM设置为推挽输出
        if(++scan_index >= 8)       scan_index = 0;     //扫描完成，重复扫描
}


/****************** LCD段码显示缓冲装载函数 ***************************/
/****************** 对第1~6数字装载显示函数 ***************************/
u8      code T_LCD_mask[4]  = {~0xc0,~0x30,~0x0c,~0x03};
u8      code T_LCD_mask4[4] = {~0x40,~0x10,~0x04,~0x01};

void    LCD_load(u8 n,u8 dat)       //n为第几个数字，为1~6，dat为要显示的数字 10us@22.1184MHZ
{       u8      i,k;
        u8      *p;
        if((n == 0) || (n > 6))     return;
        i = t_display[dat];
```

STC MCU Limited

```
    if(n <= 4)//1~4
    {       n--;
            p = LCD_buff;
    }
    else
    {       n = n - 5;
            p = &LCD_buff[4];
    }

    k = 0;
    if(i & 0x08)        k |= 0x40;                  //D
    *p = (*p & T_LCD_mask4[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x04)        k |= 0x40;                  //C
    if(i & 0x10)        k |= 0x80;                  //E
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    p++;
    k = 0;
    if(i & 0x02)        k |= 0x40;                  //B
    if(i & 0x40)        k |= 0x80;                  //G
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x01)        k |= 0x40;                  //A
    if(i & 0x20)        k |= 0x80;                  //F
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
}
```

详细的程序如下，用户也可从STC的官网www.stcmcu.com下载，下载地址为http://www.stcmcu.com/STC-DEMO-CODE/40-IO-LCD-8x6-2014-6-19.rar

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------------------------------*/
/* --- STC 1T Series MCU Demo Programme ------------------------------------------------------*/
/* --- Mobile: (86)13922805190 ---------------------------------------------------------------*/
/* --- Fax: 86-0513-55012956,55012947,55012969 -----------------------------------------------*/
/* --- Tel: 86-0513-55012928,55012929,55012966 -----------------------------------------------*/
/* --- Web: www.GXWMCU.com -------------------------------------------------------------------*/
/* --- QQ:  800003751 ------------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------*/
```

STC MCU Limited

```
#include  <intrins.h>
#include  "config.h"
#include  "timer.h"
#include  "LCD_IO16.h"

/**************     功能说明              **************
用STC115系列测试IO直接驱动段码LCD(6个8字LCD, 1/4 Dutys, 1/3 bias)。

上电后显示一个时间(时分秒).

P3.2对地接一个开关,用来进入睡眠或唤醒.
*****************************************/

/**************     本地变量声明       **************/
u8      cnt_500ms;
u8      second,minute,hour;
bit     B_Second;           //秒信号

/**************     本地函数声明       **************/
/**************  外部函数和变量声明 *****************/
extern  bit     B_2ms;

/************************** 定时器配置 ****************************/
void    Timer_config(void)
{
        TIM_InitTypeDef             TIM_InitStructure;          //结构定义
        TIM_InitStructure.TIM_Mode      = TIM_16BitAutoReload;
//指定工作模式,  TIM_16BitAutoReload,TIM_16Bit,TIM_8BitAutoReload,TIM_16BitAutoReloadNoMask

        TIM_InitStructure.TIM_Polity    = PolityLow;            //指定中断优先级, PolityHigh,PolityLow
        TIM_InitStructure.TIM_Interrupt = ENABLE;               //中断是否允许,  ENABLE或DISABLE

        TIM_InitStructure.TIM_ClkSource = TIM_CLOCK_1T;
                        //指定时钟源,    TIM_CLOCK_1T,TIM_CLOCK_12T,TIM_CLOCK_Ext

        TIM_InitStructure.TIM_ClkOut    = DISABLE;
                        //是否输出高速脉冲, ENABLE或DISABLE

        TIM_InitStructure.TIM_Value     = 65536 - (MAIN_Fosc / 500);        //初值, 节拍为500HZ
        TIM_InitStructure.TIM_Run       = ENABLE;
                        //是否初始化后启动定时器, ENABLE或DISABLE

        Timer_Inilize(Timer0,&TIM_InitStructure);               //初始化Timer0, Timer0,Timer1,Timer2
}
```

STC MCU Limited

```
/********************* 显示时间 **************************/
void	LoadRTC(void)
{
	LCD_load(1,hour/10);
	LCD_load(2,hour%10);
	LCD_load(3,minute/10);
	LCD_load(4,minute%10);
	LCD_load(5,second/10);
	LCD_load(6,second%10);
}
//========================================================================
// 函数: void  delay_ms(unsigned char ms)
// 描述: 延时函数。
// 参数: ms,要延时的ms数, 这里只支持1~255ms. 自动适应主时钟.
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//========================================================================
void  delay_ms(u8 ms)
{
	unsigned int i;
	 do
	{
		i = MAIN_Fosc / 13000;
		while(--i) ;			//14T per loop
	}while(--ms);
}

/********************* 主函数**************************/
void main(void)
{
	Init_LCD_Buffer();
	Timer_config();
	EA = 1;

	LCD_SET_2M;				//显示时分间隔:
	LCD_SET_4M;				//显示分秒间隔:
	LoadRTC();				//显示时间

	while (1)
	{
		PCON |= 0x01;		//为了省电, 进入空闲模式(电流大约2.5~3.0mA @5V),
					//由Timer0 2ms唤醒退出
```

STC MCU Limited

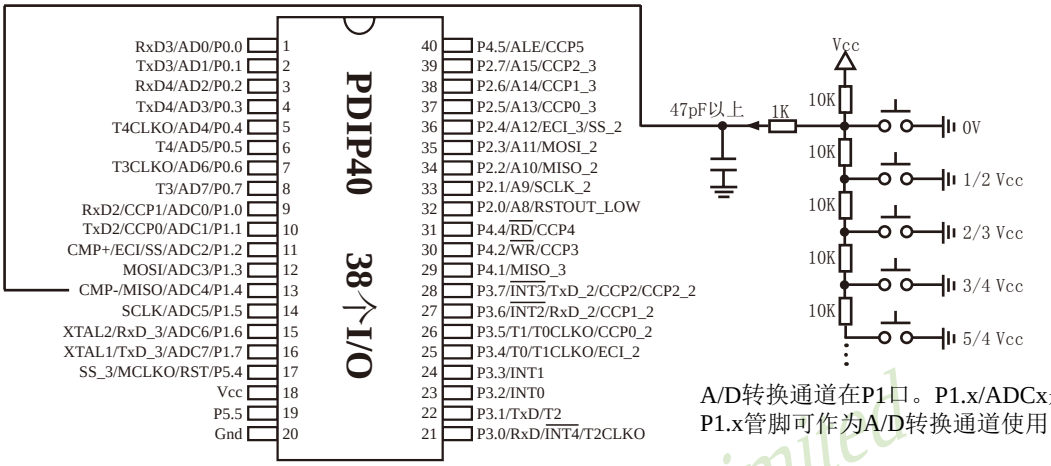
```
        _nop_();
        _nop_();
        _nop_();

        if(B_2ms)                                   //2ms节拍到
        {
            B_2ms = 0;

            if(++cnt_500ms >= 250)                  //500ms到
            {
                cnt_500ms = 0;
            //  LCD_FLASH_2M;                       //闪烁时分间隔:
            //  LCD_FLASH_4M;                       //闪烁分秒间隔:

                B_Second = ~B_Second;
                if(B_Second)
                {
                    if(++second >= 60)                          //1分钟到
                    {
                        second = 0;
                        if(++minute >= 60)                      //1小时到
                        {
                            minute = 0;
                            if(++hour >= 24)    hour = 0; //24小时到
                        }
                    }
                    LoadRTC();                                  //显示时间
                }
            }

            if(!P32)                                            //键按下，准备睡眠
            {
                LCD_CLR_2M;                                     //显示时分间隔:
                LCD_CLR_4M;                                     //显示分秒间隔:
                LCD_load(1,DIS_BLACK);
                LCD_load(2,DIS_BLACK);
                LCD_load(3,0);
                LCD_load(4,0x0F);
                LCD_load(5,0x0F);
                LCD_load(6,DIS_BLACK);

                while(!P32)         delay_ms(10);           //等待释放按键
                delay_ms(50);
                while(!P32)         delay_ms(10);           //再次等待释放按键
```

STC MCU Limited

```
                                TR0 = 0;                        //关闭定时器
                                IE0 = 0;                        //外中断0标志位
                                EX0 = 1;                        //INT0 Enable
                                IT0 = 1;                        //INT0 下降沿中断

                                P1n_push_pull(0xff);            //com和seg全部输出0
                                P2n_push_pull(0xff);            //com和seg全部输出0
                                P1 = 0;
                                P2 = 0;

                                PCON |= 0x02;                   //Sleep
                                _nop_();
                                _nop_();
                                _nop_();

                                LCD_SET_2M;                     //显示时分间隔:
                                LCD_SET_4M;                     //显示分秒间隔:
                                LoadRTC();                      //显示时间
                                TR0 = 1;                        //打开定时器
                                while(!P32)     delay_ms(10);   //等待释放按键
                                delay_ms(50);
                                while(!P32)     delay_ms(10);   //再次等待释放按键
                        }
                }
        }
}

/********************** INT0中断函数 *************************/
void INT0_int (void) interrupt INT0_VECTOR
{
        EX0 = 0;                                                //INT0 Disable
        IE0  = 0;                                               //外中断0标志位
}
```

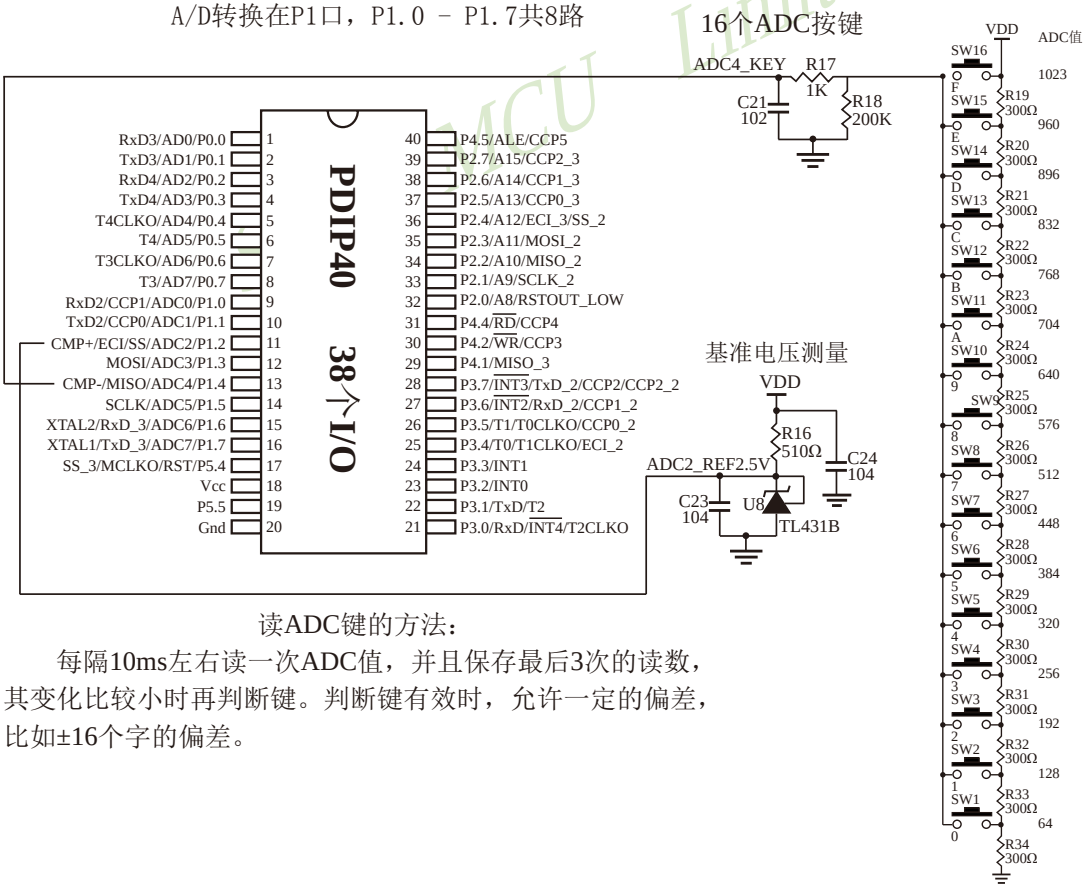
STC MCU Limited

4.23 A/D做按键扫描应用线路图



A/D转换在P1口，P1.0 – P1.7共8路

A/D转换通道在P1口。P1.x/ADCx是指P1.x管脚可作为A/D转换通道使用。



读ADC键的方法：

每隔10ms左右读一次ADC值，并且保存最后3次的读数，其变化比较小时再判断键。判断键有效时，允许一定的偏差，比如±16个字的偏差。

4.24 STC15系列单片机I/O口软件模拟I²C接口的测试程序

4.24.1 STC15系列单片机I/O口软件模拟I2C接口的主机模式

```
;/*---------------------------------------------------------------------------*/
;/* --- STC MCU Limited. ---------- ----------------------------------*/
;/* --- STC 1T Series MCU Simulate I2C Master Demo ----------*/
;/* --- Mobile: (86)13922829991 ----------------------------------------*/
;/* --- Fax: 86-755-82944243 -------------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------------------*/

SCL     BIT     P1.0
SDA     BIT     P1.1

;----------------------------

        ORG     0000H

        MOV     TMOD,   #20H            ;初始化串口为(9600,n,8,1)
        MOV     SCON,   #5AH
        MOV     A,      #-5             ;-18432000/12/32/9600
        MOV     TH1,    A
        MOV     TL1,    A
        SETB    TR1
MAIN:
        CALL    UART_RXDATA             ;接收下一个串口数据
        MOV     R0,     A               ;临时保存到R0
                                        ;读取I2C设备IDATA 80H的数据
        CALL    I2C_START               ;开始读取
        MOV     A,      #01H
        CALL    I2C_TXBYTE              ;发送地址数据+读信号
        CALL    I2C_RXACK               ;接收ACK
        CALL    I2C_RXBYTE              ;接收数据
        SETB    C
        CALL    I2C_TXACK               ;发送NAK
        CALL    I2C_STOP                ;读取完成

        CALL    UART_TXDATA             ;将读到的数据发送到串口
                                        ;将R0的数据写入I2C设备IDATA 80H
        CALL    I2C_START               ;开始写
        MOV     A,      #00H
        CALL    I2C_TXBYTE              ;发送地址数据+写信号
        CALL    I2C_RXACK               ;接收ACK
        MOV     A,      R0
```

STC MCU Limited

```
        CALL    I2C_TXBYTE              ;写数据
        CALL    I2C_RXACK               ;接收ACK
        CALL    I2C_STOP                ;写完成

        JMP     MAIN

;----------------------------
;等待串口数据
;----------------------------
UART_RXDATA:
        JNB     RI,     $               ;等待接收完成标志
        CLR     RI                      ;清除标志
        MOV     A,      SBUF            ;保存数据
        RET

;----------------------------
;发送串口数据
;----------------------------
UART_TXDATA:
        JNB     TI,     $               ;等待上一个数据发送完成
        CLR     TI                      ;清除标志
        MOV     SBUF,   A               ;发送数据
        RET

;----------------------------
;发送I2C起始信号
;----------------------------
I2C_START:
        CLR     SDA                     ;数据线下降沿
        CALL    I2C_DELAY               ;延时
        CLR     SCL                     ;时钟->低
        CALL    I2C_DELAY               ;延时
        RET

;----------------------------
;发送I2C停止信号
;----------------------------
I2C_STOP:
        CLR     SDA
        SETB    SCL                     ;时钟->高
        CALL    I2C_DELAY               ;延时
        SETB    SDA                     ;数据线上升沿
        CALL    I2C_DELAY               ;延时
        RET

;----------------------------
;发送ACK/NAK信号
;----------------------------
```

STC MCU Limited

```
I2C_TXACK:
        MOV     SDA,    C               ;送ACK数据
        SETB    SCL                     ;时钟->高
        CALL    I2C_DELAY               ;延时
        CLR     SCL                     ;时钟->低
        CALL    I2C_DELAY               ;延时
        SETB    SDA                     ;发送完成
        RET

;-----------------------------
;接收ACK/NAK信号
;-----------------------------
I2C_RXACK:
        SETB    SDA                     ;准备读数据
        SETB    SCL                     ;时钟->高
        CALL    I2C_DELAY               ;延时
        MOV     C,      SDA             ;读取ACK信号
        CLR     SCL                     ;时钟->低
        CALL    I2C_DELAY               ;延时
        RET

;-----------------------------
;接收一字节数据
;-----------------------------
I2C_TXBYTE:
        MOV     R7,     #8              ;8位计数
TXNEXT:
        RLC     A                       ;移出数据位
        MOV     SDA,    C               ;数据送数据口
        SETB    SCL                     ;时钟->高
        CALL    I2C_DELAY               ;延时
        CLR     SCL                     ;时钟->低
        CALL    I2C_DELAY               ;延时
        DJNZ    R7,     TXNEXT          ;送下一位
        RET

;-----------------------------
;发送一字节数据
;-----------------------------
I2C_RXBYTE:
        MOV     R7,     #8              ;8位计数
RXNEXT:
        SETB    SCL                     ;时钟->高
        CALL    I2C_DELAY               ;延时
        MOV     C,      SDA
        RLC     A
        CLR     SCL                     ;时钟->低
        CALL    I2C_DELAY               ;延时
```

STC MCU Limited

```
        DJNZ    R7,     RXNEXT          ;收下一位
        RET

;------------------------------

I2C_DELAY:                              ;6
        PUSH    0                       ;4
        MOV     R0,     #1              ;2 6(200K) 1(400K)  [18'432'000/400'000=46]
        DJNZ    R0,     $               ;4
        POP     0                       ;3
        RET                             ;4

;------------------------------

        END
```

STC MCU Limited

4.24.2 STC15系列单片机I/O口软件模拟I2C接口的从机模式

```
;/*---------------------------------------------------------------------------*/
;/* --- STC MCU Limited. ---------- ----------------------------------*/
;/* --- STC 1T Series MCU Simulate I2C Slave Demo ----------------*/
;/* --- 研发顾问QQ：800003751----------------------------------------*/
;/* --- Fax: 86-755-82944243 -----------------------------------------*/
;/* --- Tel: 86-755-82948412 -----------------------------------------*/
;/* --- Web: www.STCMCU.com ------------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------------------*/

SCL     BIT     P1.0
SDA     BIT     P1.1

;----------------------------

        ORG     0

RESET:
        SETB    SCL
        SETB    SDA

        CALL    I2C_WAITSTART           ;等待起始信号
        CALL    I2C_RXBYTE              ;接收地址数据
        CLR     C
        CALL    I2C_TXACK               ;回应ACK
        SETB    C                       ;读/写 IDATA[80H - FFH]
        RRC     A                       ;读/写位->C
        MOV     R0,     A               ;地址送入R0
        JC      READDATA                ;C=1(读) C=0(写)
WRITEDATA:
        CALL    I2C_RXBYTE              ;接收数据
        MOV     @R0,    A               ;写入IDATA
        INC     R0                      ;地址+1
        CLR     C
        CALL    I2C_TXACK               ;回应ACK
        CALL    I2C_WAITSTOP            ;等待停止信号
        JMP     RESET

READDATA:
        MOV     A,      @R0
        INC     R0
        CALL    I2C_TXBYTE              ;发送IDATA数据
        CALL    I2C_RXACK               ;接收ACK
        CALL    I2C_WAITSTOP            ;等待停止信号
        JMP     RESET
```

STC MCU Limited

```
;-----------------------------
;等待起始信号
;-----------------------------
I2C_WAITSTART:
        JNB     SCL,    $               ;等待时钟->高
        JB      SDA,    $               ;等待数据线下降沿
        JB      SCL,    $               ;等待时钟->低
        RET

;-----------------------------
;等待结束信号
;-----------------------------
I2C_WAITSTOP:
        JNB     SCL,    $               ;等待时钟->高
        JNB     SDA,    $               ;等待数据线上升沿
        RET

;-----------------------------
;发送ACK/NAK信号
;-----------------------------
I2C_TXACK:
        MOV     SDA,    C               ;送ACK数据
        JNB     SCL,    $               ;等待时钟->高
        JB      SCL,    $               ;等待时钟->低
        SETB    SDA                     ;发送完成
        RET

;-----------------------------
;接收ACK/NAK信号
;-----------------------------
I2C_RXACK:
        SETB    SDA                     ;准备读数据
        JNB     SCL,    $               ;等待时钟->高
        MOV     C,      SDA             ;读取ACK信号
        JB      SCL,    $               ;等待时钟->低
        RET

;-----------------------------
;接收一字节数据
;-----------------------------
I2C_RXBYTE:
        MOV     R7,     #8              ;8位计数
RXNEXT:
        JNB     SCL,    $               ;等待时钟->高
        MOV     C,      SDA             ;读取数据口
        RLC     A                       ;保存数据
        JB      SCL,    $               ;等待时钟->低
        DJNZ    R7,     RXNEXT          ;收下一位
        RET
```

STC MCU Limited

```
;-----------------------------
;发送一字节数据
;-----------------------------
I2C_TXBYTE:
        MOV     R7,     #8              ;8位计数
TXNEXT:
        RLC     A                       ;移出数据位
        MOV     SDA,    C               ;数据送数据口
        JNB     SCL,    $               ;等待时钟->高
        JB      SCL,    $               ;等待时钟->低
        DJNZ    R7,     TXNEXT          ;送下一位
        RET

;-----------------------------

        END
```

STC MCU Limited

第5章 指令系统

5.1 寻址方式

寻址方式是每一种计算机的指令集中不可缺少的部分。寻址方式规定了数据的来源和目的地。对不同的程序指令，来源和目的地的规定也会不同。在STC单片机中的寻址方式可概括为：

- 立即寻址
- 直接寻址
- 间接寻址
- 寄存器寻址
- 相对寻址
- 变址寻址
- 位寻址

5.1.1 立即寻址

立即寻址也称立即数，它是在指令操作数中直接给出参加运算的操作数，其指令格式如下：

如：MOV A, #70H

这条指令的功能是将立即数70H传送到累加器A中

5.1.2 直接寻址

在直接寻址方式中，指令操作数域给出的是参加运算操作数地址。直接寻址方式只能用来表示特殊功能寄存器、内部数据寄存器和位地址空间。其中特殊功能寄存器和位地址空间只能用直接寻址方式访问。

如：ANL 70H, #48H

表示70H单元中的数与立即数48H相“与”，结果存放在70H单元中。其中70H为直接地址，表示内部数据存储器RAM中的一个单元。

5.1.3 间接寻址

间接寻址采用R0或R1前添加“@”符号来表示。例如，假设R1中的数据是40H，内部数据存储器40H单元所包含的数据为55H，那么如下指令：

MOV A, @R1

把数据55H传送到累加器。

STC MCU Limited

5.1.4 寄存器寻址

寄存器寻址是对选定的工作寄存器R7~R0、累加器A、通用寄存器B、地址寄存器和进位C中的数进行操作。其中寄存器R7~R0由指令码的低3位表示，ACC、B、DPTR及进位位C隐含在指令码中。因此，寄存器寻址也包含一种隐含寻址方式。

寄存器工作区的选择由程序状态字寄存器PSW中的RS1、RS0来决定。指令操作数指定的寄存器均指当前工作区中的寄存器。

如：INC　R0　　;(R0)+1 → R0

5.1.5 相对寻址

相对寻址是将程序计数器PC中的当前值与指令第二字节给出的数相加，其结果作为转移指令的转移地址。转移地址也称为转移目的地址，PC中的当前值称为基地址，指令第二字节给出的数称为偏移量。由于目的地址是相对于PC中的基地址而言，所以这种寻址方式称为相对寻址。偏移量为带符号的数，所能表示的范围为+127～–128。这种寻址方式主要用于转移指令。

如：JC　80H　;C=1 跳转

表示若进位位C为0，则程序计数器PC中的内容不改变，即不转移。若进位位C为1，则以PC中的当前值为基地址，加上偏移量80H后所得到的结果作为该转移指令的目的地址。

5.1.6 变址寻址

在变址寻址方式中，指令操作数指定一个存放变址基值的变址寄存器。变址寻址时，偏移量与变址基值相加，其结果作为操作数的地址。变址寄存器有程序计数器PC和地址寄存器DPTR.

如：MOVC　A,　@A+DPTR

表示累加器A为偏移量寄存器，其内容与地址寄存器DPTR中的内容相加，其结果作为操作数的地址，取出该单元中的数送入累加器A。

5.1.7 位寻址

位寻址是指对一些内部数据存储器RAM和特殊功能寄存器进行位操作时的寻址。在进行位操作时，借助于进位位C作为位操作累加器，指令操作数直接给出该位的地址，然后根据操作码的性质对该位进行位操作。位地址与字节直接寻址中的字节地址形式完全一样，主要由操作码加以区分，使用时应注意。

如：MOV　C,　20H　　；片内位单元位操作型指令

STC MCU Limited

5.2 完整指令集对照表(与传统8051对照)

——共111条指令，每条指令的详细执行时间

----与普通8051指令代码完全兼容，但执行的时间效率大幅提升
----其中INC DPTR和MUL AB指令的执行速度大幅提升24倍
----共有22条指令，一个时钟就可以执行完成，平均速度快8~12倍

如果按功能分类，STC15系列单片机指令系统可分为：
1. 算术操作类指令；
2. 逻辑操作类指令；
3. 数据传送类指令；
4. 布尔变量操作类指令；
5. 控制转移类指令。

按功能分类的指令系统表如下表所示。

STC MCU Limited

指令执行速度效率提升总结(新15系列)：

指令系统共包括111条指令，其中：

执行速度快24倍的	共2条
执行速度快12倍的	共28条
执行速度快8倍的	共19条
执行速度快6倍的	共40条
执行速度快4.8倍的	共8条
执行速度快4倍的	共14条

根据对指令的使用频率分析统计，STC15系列 1T 的8051单片机比普通的8051单片机在同样的工作频率下运行速度提升了8～12倍。

指令执行时钟数统计（供参考）(新15系列)：

指令系统共包括111条指令，其中：

1个时钟就可执行完成的指令	共22条
2个时钟就可执行完成的指令	共37条
3个时钟就可执行完成的指令	共31条
4个时钟就可执行完成的指令	共12条
5个时钟就可执行完成的指令	共8条
6个时钟就可执行完成的指令	共1条

STC15系列将111条指令全部执行完一遍所需的时钟为283个时钟，而传统8051单片机将111条指令全部执行一遍要1944个时钟。可见与传统8051相比较，STC新15系列的指令执行速度大幅提升，平均速度快8~12倍。

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.

算术操作类指令

助记符		功能说明	字节数	传统8051单片机所需时钟	STC15系列单片机所需时钟（采用STC-Y5超高速 1T 8051 内核）	效率提升
ADD	A，Rn	寄存器内容加到累加器	1	12	1	12倍
ADD	A，direct	直接地址单元中的数据加到累加器	2	12	2	6倍
ADD	A，@Ri	间接RAM中的数据加到累加器	1	12	2	6倍
ADD	A，#data	立即数加到累加器	2	12	2	6倍
ADDC	A，Rn	寄存器带进位加到累加器	1	12	1	12倍
ADDC	A，direct	直接地址单元的内容带进位加到累加器	2	12	2	6倍
ADDC	A，@Ri	间接RAM内容带进位加到累加器	1	12	2	6倍
ADDC	A，#data	立即数带进位加到累加器	2	12	2	6倍
SUBB	A，Rn	累加器带借位减寄存器内容	1	12	1	6倍
SUBB	A，direct	累加器带借位减直接地址单元的内容	2	12	2	6倍
SUBB	A，@Ri	累加器带借位减间接RAM中的内容	1	12	2	6倍
SUBB	A，#data	累加器带借位减立即数	2	12	2	6倍
INC	A	累加器加1	1	12	1	12倍
INC	Rn	寄存器加1	1	12	2	6倍
INC	direct	直接地址单元加1	2	12	3	4倍
INC	@Ri	间接RAM单元加1	1	12	3	4倍
DEC	A	累加器减1	1	12	1	12倍
DEC	Rn	寄存器减1	1	12	2	6倍
DEC	direct	直接地址单元减1	2	12	3	4倍
DEC	@Ri	间接RAM单元减1	1	12	3	4倍
INC	DPTR	地址寄存器DPTR加1	1	24	1	24倍
MUL	AB	A乘以B	1	48	2	24倍
DIV	AB	A除以B	1	48	6	8倍
DA	A	累加器十进制调整	1	12	3	4倍

STC MCU Limited

逻辑操作类指令

助记符		功能说明	字节数	传统8051单片机所需时钟	STC15系列单片机所需时钟（采用STC-Y5超高速 1T 8051 内核）	效率提升
ANL	A，Rn	累加器与寄存器相“与”	1	12	1	12倍
ANL	A，direct	累加器与直接地址单元相“与”	2	12	2	6倍
ANL	A，@Ri	累加器与间接RAM单元相“与”	1	12	2	6倍
ANL	A，#data	累加器与立即数相“与”	2	12	2	6倍
ANL	direct， A	直接地址单元与累加器相“与”	2	12	3	4倍
ANL	direct，#data	直接地址单元与立即数相“与”	3	24	3	8倍
ORL	A， Rn	累加器与寄存器相“或”	1	12	1	12倍
ORL	A，direct	累加器与直接地址单元相“或”	2	12	2	6倍
ORL	A, @Ri	累加器与间接RAM单元相“或”	1	12	2	6倍
ORL	A，# data	累加器与立即数相“或”	2	12	2	6倍
ORL	direct， A	直接地址单元与累加器相“或”	2	12	3	4倍
ORL	direct， #data	直接地址单元与立即数相“或”	3	24	3	8倍
XRL	A， Rn	累加器与寄存器相“异或”	1	12	1	12倍
XRL	A， direct	累加器与直接地址单元相“异或”	2	12	2	6倍
XRL	A， @Ri	累加器与间接RAM单元相“异或”	1	12	2	6倍
XRL	A， # data	累加器与立即数相“异或”	2	12	2	6倍
XRL	direct， A	直接地址单元与累加器相“异或”	2	12	3	4倍
XRL	direct，#data	直接地址单元与立即数相“异或”	3	24	3	8倍
CLR	A	累加器清“ 0”	1	12	1	12倍
CPL	A	累加器求反	1	12	1	12倍
RL	A	累加器循环左移	1	12	1	12倍
RLC	A	累加器带进位位循环左移	1	12	1	12倍
RR	A	累加器循环右移	1	12	1	12倍
RRC	A	累加器带进位位循环右移	1	12	1	12倍
SWAP	A	累加器内高低半字节交换	1	12	1	12倍

STC MCU Limited

数据传送类指令

助记符		功能说明	字节数	传统8051单片机所需时钟	STC15系列单片机所需时钟（采用STC-Y5超高速 1T 8051 内核）	效率提升
MOV	A, Rn	寄存器内容送入累加器	1	12	1	12倍
MOV	A, direct	直接地址单元中的数据送入累加器	2	12	2	6倍
MOV	A, @Ri	间接RAM中的数据送入累加器	1	12	2	6倍
MOV	A, #data	立即数送入累加器	2	12	2	6倍
MOV	Rn, A	累加器内容送入寄存器	1	12	1	12倍
MOV	Rn, direct	直接地址单元中的数据送入寄存器	2	24	3	8倍
MOV	Rn, #data	立即数送入寄存器	2	12	2	6倍
MOV	direct, A	累加器内容送入直接地址单元	2	12	2	6倍
MOV	direct, Rn	寄存器内容送入直接地址单元	2	24	2	12倍
MOV	direct, direct	直接地址单元中的数据送入另一个直接地址单元	3	24	3	8倍
MOV	direct, @Ri	间接RAM中的数据送入直接地址单元	2	24	3	8倍
MOV	direct, #data	立即数送入直接地址单元	3	24	3	8倍
MOV	@Ri, A	累加器内容送间接RAM单元	1	12	2	6倍
MOV	@Ri, direct	直接地址单元数据送入间接RAM单元	2	24	3	8倍
MOV	@Ri, #data	立即数送入间接RAM单元	2	12	2	6倍
MOV	DPTR,#data16	16位立即数送入数据指针	3	24	3	8倍
MOVC	A, @A+DPTR	以DPTR为基地址变址寻址单元中的数据送入累加器	1	24	5	4.8倍
MOVC	A, @A+PC	以PC为基地址变址寻址单元中的数据送入累加器	1	24	4	6倍
MOVX	A, @Ri	将逻辑上在片外、物理上在片内的扩展RAM(8位地址)的内容送入累加器A中，读操作	1	24	3	8倍
MOVX	@Ri, A	将累加器A的内容送入逻辑上在片外、物理上在片内的扩展RAM(8位地址)中，写操作	1	24	4	8倍
MOVX	A, @DPTR	将逻辑上在片外、物理上在片内的扩展RAM(16位地址)的内容送入累加器A中，读操作	1	24	2	12倍
MOVX	@DPTR, A	将累加器A的内容送入逻辑上在片外、物理上在片内的扩展RAM(16位地址)中，写操作	1	24	3	8倍
MOVX	A, @Ri	将逻辑上在片外、物理上也在片外的扩展RAM(8位地址)的内容送入累加器A中，读操作	1	24	5xN+2 N的取值见下列说明	*Note1
MOVX	@Ri, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(8位地址)中，写操作	1	24	5×N+3 N的取值见下列说明	*Note1
MOVX	A, @DPTR	将逻辑上在片外、物理上也在片外的扩展RAM(16位地址)的内容送入累加器A中，读操作	1	24	5×N+1 N的取值见下列说明	*Note1
MOVX	@DPTR, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(16位地址)中，写操作	1	24	5×N+2 N的取值见下列说明	*Note1
PUSH	direct	直接地址单元中的数据压入堆栈	2	24	3	8倍
POP	direct	栈底数据弹出送入直接地址单元	2	24	2	12倍
XCH	A, Rn	寄存器与累加器交换	1	12	2	6倍
XCH	A,direct	直接地址单元与累加器交换	2	12	3	4倍
XCH	A, @Ri	间接RAM与累加器交换	1	12	3	4倍
XCHD	A, @Ri	间接RAM的低半字节与累加器交换	1	12	3	4倍

当EXRTS[1:0] = [0,0]时，表中N=1;
当EXRTS[1:0] = [0,1]时，表中N=2;
当EXRTS[1:0] = [1,0]时，表中N=4;
当EXRTS[1:0] = [1,1]时，表中N=8.　　EXRTS[1：0]为寄存器BUS_SPEED中的B1,B0位

布尔变量操作类指令

助记符		功能说明	字节数	传统8051单片机所需时钟	STC15系列单片机所需时钟(采用STC-Y5超高速 1T 8051 内核)	效率提升
CLR	C	清零进位位	1	12	1	12倍
CLR	bit	清0直接地址位	2	12	3	4倍
SETB	C	置1进位位	1	12	1	12倍
SETB	bit	置1直接地址位	2	12	3	4倍
CPL	C	进位位求反	1	12	1	12倍
CPL	bit	直接地址位求反	2	12	3	4倍
ANL	C, bit	进位位和直接地址位相“与”	2	24	2	12倍
ANL	C, /bit	进位位和直接地址位的反码相“与”	2	24	2	12倍
ORL	C, bit	进位位和直接地址位相“或”	2	24	2	12倍
ORL	C, /bit	进位位和直接地址位的反码相“或”	2	24	2	12倍
MOV	C, bit	直接地址位送入进位位	2	12	2	12倍
MOV	bit, C	进位位送入直接地址位	2	24	3	8倍
JC	rel	进位位为1则转移	2	24	3	8倍
JNC	rel	进位位为0则转移	2	24	3	8倍
JB	bit, rel	直接地址位为1则转移	3	24	5	4.8倍
JNB	bit, rel	直接地址位为0则转移	3	24	5	4.8倍
JBC	bit, rel	直接地址位为1则转移，该位清0	3	24	5	4.8倍

STC MCU Limited

控制转移类指令

助记符	功能说明	字节数	传统8051单片机所需时钟	STC15系列单片机所需时钟(采用STC-Y5超高速 1T 8051 内核)	效率提升
ACALL addr11	绝对（短）调用子程序	2	24	4	6倍
LCALL addr16	长调用子程序	3	24	4	6倍
RET	子程序返回	1	24	4	6倍
RETI	中断返回	1	24	4	6倍
AJMP addr11	绝对（短）转移	2	24	3	8倍
LJMP addr16	长转移	3	24	4	6倍
SJMP re1	相对转移	2	24	3	8倍
JMP @A+DPTR	相对于DPTR的间接转移	1	24	5	4.8倍
JZ re1	累加器为零转移	2	24	4	6倍
JNZ re1	累加器非零转移	2	24	4	6倍
CJNE A，direct，re1	累加器与直接地址单元比较，不相等则转移	3	24	5	4.8倍
CJNE A，#data，re1	累加器与立即数比较，不相等则转移	3	24	4	6倍
CJNE Rn，#data，re1	寄存器与立即数比较，不相等则转移	3	24	4	6倍
CJNE @Ri，#data，re1	间接RAM单元与立即数比较，不相等则转移	3	24	5	4.8倍
DJNZ Rn，re1	寄存器减1，非零转移	2	24	4	6倍
DJNZ direct，re1	直接地址单元减1，非零转移	3	24	5	4.8倍
NOP	空操作	1	12	1	12倍

本次指令系统总结更新于2011-10-17日止

STC-MCU Limited

5.3 传统8051单片机指令定义详解(中文&English)

5.3.1 传统8051单片机指令定义详解

ACALL addr 11

功能： 绝对调用

说明： ACALL指令实现无条件调用位于addr11参数所表示地址的子例程。在执行该指令时，首先将PC的值增加2，即使得PC指向ACALL的下一条指令，然后把16位PC的低8位和高8位依次压入栈，同时把栈指针两次加1。然后，把当前PC值的高5位、ACALL指令第1字节的7~5位和第2字节组合起来，得到一个16 位目的地址，该地址即为即将调用的子例程的入口地址。要求该子例程的起始地址必须与紧随ACALL之后的指令处于同1个2KB的程序存储页中。ACALL指令在执行时不会改变各个标志位。

举例： SP的初始值为07H，标号SUBRTN位于程序存储器的0345H地址处，如果执行位于地址0123H处的指令：

ACALL SUBRTN

那么SP变为09H，内部RAM地址08H和09H单元的内容分别为25H和01H，PC值变为0345H。

指令长度(字节)： 2

执行周期： 2

二进制编码：

a10	a9	a8	1	0	0	1	0	a7	a6	a5	a4	a3	a2	a1	a0

注意：a10 a9 a8是11位目标地址addr11的A10~A8位，a7 a6 a5 a4 a3 a2 a1 a0是addr11的A7~A0位。

操作： ACALL

(PC)← **(PC)+ 2**

(SP)←**(SP) + 1**

((sP)) ← **(PC_{7-0})**

(SP)←**(SP) + 1**

((SP))←**(PC_{15-8})**

(PC_{10-0})← 页码地址

STC MCU Limited

ADD　A, <src-byte>

功能： 加法

说明： ADD指令可用于完成把src-byte所表示的源操作数和累加器*A*的当前值相加。并将结果置于累加器*A*中。根据运算结果，若第7位有进位则置进位标志为1，否则清零；若第3位有进位则置辅助进位标志为1，否则清零。如果是无符号整数相加则进位置位，显示当前运算结果发生溢出。

如果第6位有进位生成而第7位没有，或第7位有进位生成而第6位没有，则置OV为1，否则OV被清零。在进行有符号整数的相加运算的时候，OV置位表示两个正整数之和为一负数，或是两个负整数之和为一正数。

本类指令的源操作数可接受4种寻址方式：寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。

举例： 假设累加器*A*中的数据为0C3H(000011B)，R0的值为0AAH(10101010B)。执行如下指令：

```
ADD    A, R0
```

累加器*A*中的结果为6DH(01101101B)，辅助进位标志AC被清零，进位标志C和溢出标志OV被置1。

STC MCU Limited

ADD　A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

0 0 1 0	1 r r r

操作： ADD

(A)←**(A) + (Rn)**

ADD　A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

0 0 1 0	0 1 0 1	direct address

操作： ADD

(A)←**(A) + (direct)**

ADD　A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

0 0 1 0	0 1 1 i

操作： ADD

(A)←**(A) + ((Ri))**

ADD　A, #data

指令长度(字节)：2

执行周期：1

二进制编码：

0 0 1 0	0 1 0 0	immediate data

操作：ADD

(A)←**(A) + #data**

ADDC　A, <src-byte>

功能：带进位的加法。

说明：执行ADDC指令时，把src-byte所代表的源操作数连同进位标志一起加到累加器*A*上，并将结果置于累加器*A*中。根据运算结果，若在第7位有进位生成，则将进位标志置1，否则清零；若在第3位有进位生成，则置辅助进位标志为1，否则清零。如果是无符号数整数相加，进位的置位显示当前运算结果发生溢出。

如果第6位有进位生成而第7位没有，或第7位有进位生成而第6位没有，则将OV置1，否则将OV清零。在进行有符号整数相加运算的时候，OV置位，表示两个正整数之和为一负数，或是两个负整数之和为一正数。

本类指令的源操作数允许4种寻址方式：寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。

举例：假设累加器*A*中的数据为0C3H(11000011B)，*R*0的值为0AAH(10101010B)，进位标志为1，执行如下指令：

ADDC A,R0

累加器*A*中的结果为6EH(01101110B)，辅助进位标志AC被清零，进位标志C和溢出标志OV被置1。

ADDC　A, Rn

指令长度(字节)：1

执行周期：1

二进制编码：

0 0 1 1	1 r r r

操作：ADDC

(A)←**(A) + (C) + (Rn)**

ADDC　A,direct

指令长度(字节)：2

执行周期：1

二进制编码：

0 0 1 1	0 1 0 1	direct address

操作：ADDC

(A)←**(A) + (C) + (direct)**

STC MCU Limited

ADDC　A, @Ri

指令长度(字节)：1

执行周期：1

二进制编码：

0 0 1 1	0 1 1 i

操作：ADDC

(A)←**(A) + (C) + ((Ri))**

ADDC　A, #data

指令长度(字节)：2

执行周期：1

二进制编码：

0 0 1 1	0 1 0 0	immediate data

操作：ADDC

(A)←**(A) + (C) + #data**

AJMP　addr 11

功能：绝对跳转

说明：AJMP指令用于将程序转到相应的目的地址去执行，该地址在程序执行过程之中产生，由PC值（两次递增之后）的高5位、操作码的7~5位和指令的第2字节连接形成。要求跳转的目的地址和AJMP指令的后一条指令的第1字节位于同一2KB的程序存储页内。

举例：假设标号JMPADR位于程序存储器的0123H，指令

AJMP　JMPADR

位于0345H，执行完该指令后PC值变为0123H。

指令长度(字节)：2

执行周期：2

二进制编码：

a10 a9 a8 0	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

注意：目的地址的A10－A8＝a10~a8，A7－A0＝a7~a0

操作：AJMP

(PC)← **(PC)+ 2**

($PC_{10\text{-}0}$)← **page address**

STC MCU Limited

ANL　<dest-byte> , <src-byte>

功能： 对字节变量进行逻辑与运算

说明： ANL指令将由<dest-byte>和<src-byte>所指定的两个字节变量逐位进行逻辑与运算，并将运算结果存放在<dest-byte>所指定的目的操作数中。该指令的执行不会影响标志位。

两个操作数组合起来允许6种寻址模式。当目的操作数为累加器时，源操作数允许寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。当目的操作数是直接地址时，源操作数可以是累加器或立即数。

注意：当该指令用于修改输出端口时，读入的原始数据来自于输出数据的锁存器而非输入引脚。

举例： 如果累加器的内容为0C3H(11000011B)，寄存器0的内容为55H(010101011B)，那么指令:

```
ANL   A,R0
```

执行结果是累加器的内容变为41H(01000001H)。

当目的操作数是可直接寻址的数据时，ANL指令可用来把任何RAM单元或者硬件寄存器中的某些位清零。屏蔽字节将决定哪些位将被清零。屏蔽字节可能是常数，也可能是累加器在计算过程中产生。如下指令:

```
ANL   Pl, #01110011B
```

将端口1的位7、位3和位2清零。

STC MCU Limited

ANL　A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

0 1 0 1	1 r r r

操作： ANL

(A)←**(A)** ∧ (Rn)

ANL　A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

0 1 0 1	0 1 0 1	direct address

操作： ANL

(A)←**(A)** ∧ (direct)

ANL　A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

0 1 0 1	0 1 1 i

操作： ANL

(A)←**(A)** ∧ ((Ri))

ANL　A, #data

指令长度(字节)：2

执行周期：1

二进制编码：

0 1 0 1	0 1 0 0	immediate data

操作：ANL
(A)←**(A)** ∧ #data

ANL　direct, A

指令长度(字节)：2

执行周期：1

二进制编码：

0 1 0 1	0 0 1 0	direct address

操作：ANL
(direct)←**(direct)** ∧ (A)

ANL　direct, #data

指令长度(字节)：3

执行周期：2

二进制编码：

0 1 0 1	0 0 1 1	direct address	immediate data

操作：ANL
(direct)←**(direct)** ∧ #data

ANL　C , <src-bit>

功能：对位变量进行逻辑与运算

说明：如果src-bit表示的布尔变量为逻辑0，清零进位标志位；否则，保持进位标志的当前状态不变。在汇编语言程序中，操作数前面的“/”符号表示在计算时需要先对被寻址位取反，然后才作为源操作数，但源操作数本身不会改变。该指令在执行时不会影响其他各个标志位。

源操作数只能采取直接寻址方式。

举例：下面的指令序列当且仅当P1.0＝1、ACC.7＝1和OV＝0时，将进位标志C置1:

```
MOV   C, P1.0          ;LOAD CARRY WITH INPUT PIN STATE
ANL   C, ACC.7         ;AND CARRY WITH ACCUM. BIT.7
ANL   C, /OV           ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL　C, bit

指令长度(字节)：2

执行周期：2

二进制编码：

1 0 0 0	0 0 1 0	bit address

操作：ANL
(C) ← **(C)** ∧ (bit)

STC MCU Limited

ANL　C, /bit

指令长度(字节)：2

执行周期：2

二进制编码：

1 0 1 1	0 0 0 0	bit address

操作：ANL

(C)←**(C)** ∧ $\overline{(bit)}$

CJNE　<dest-byte>, <src-byte>, rel

功能：若两个操作数不相等则转移

说明：　CJNE首先比较两个操作数的大小，如果二者不等则程序转移。目标地址由位于CJNE指令最后1个字节的有符号偏移量和PC的当前值（紧邻CJNE的下一条指令的地址）相加而成。如果目标操作数作为一个无符号整数，其值小于源操作数对应的无符号整数，那么将进位标志置1，否则将进位标志清零。但操作数本身不会受到影响。

<dest-byte>和<src-byte>组合起来，允许4种寻址模式。累加器*A*可以与任何可直接寻址的数据或立即数进行比较，任何间接寻址的RAM单元或当前工作寄存器都可以和立即常数进行比较。

举例：设累加器*A*中值为34H，*R7*包含的数据为56H。如下指令序列：

```
                CJNE    R7,#60H, NOT-EQ
;               . . .   . . . . .       ; R7 = 60H.
NOT_EQ:         JC      REQ_LOW         ; IF R7 < 60H.
;               . . .   . . . . .       ; R7 > 60H.
```

的第1条指令将进位标志置1，程序跳转到标号NOT_EQ处。接下去，通过测试进位标志，可以确定*R7*是大于60H还是小于60H。

假设端口1的数据也是34H，那么如下指令：

```
WAIT:   CJNE   A,P1,WAIT
```

清除进位标志并继续往下执行，因为此时累加器的值也为34H，即和P1口的数据相等。（如果*P*l端口的数据是其他的值，那么程序在此不停地循环，直到*P*1端口的数据变成34H为止。）

CJNE　A, direct, rel

指令长度(字节)：3

执行周期：2

二进制编码：

1 0 1 1	0 1 0 1	direct address	rel. address

操作：(PC) ← **(PC) + 3**

IF (A) < > *(direct)*

THEN

　　(PC) ← **(PC) +** *relative offset*

IF (A) < *(direct)*

THEN

　　(C) ← **1**

ELSE

　　(C) ← **0**

STC MCU Limited

CJNE　A, #data, rel

指令长度(字节)：3

执行周期：2

二进制编码：

1 0 1 1	0 1 0 1	immediata data	rel. address

操作：(PC) ← **(PC) + 3**
IF (A) < > *(data)*
THEN
　　(PC) ← **(PC)** + *relative offset*
IF (A) < *(data)*
THEN
　　(C) ← **1**
ELSE
　　(C) ← **0**

CJNE　Rn, #data, rel

指令长度(字节)：3

执行周期：2

二进制编码：

1 0 1 1	1 r r r	immediata data	rel. address

操作：(PC) ← **(PC) + 3**
IF (Rn) < > *(data)*
THEN
　　(PC) ← **(PC)** + *relative offset*
IF (Rn) < *(data)*
THEN
　　(C) ← **1**
ELSE
　　(C) ← **0**

CJNE　@Ri,#data,rel

指令长度(字节)：3

执行周期：2

二进制编码：

1 0 1 1	0 1 1 i	immediate data	rel. address

操作：(PC) ← **(PC) + 3**
IF ((Ri)) < > *(data)*
THEN
　　(PC) ← **(PC)** + *relative offset*
IF ((Ri)) < *(data)*
THEN
　　(C) ← **1**
ELSE
　　(C) ← **0**

STC MCU Limited

CLR A

功能：清除累加器

说明：该指令用于将累加器*A*的所有位清零，不影响标志位。

举例：假设累加器*A*的内容为5CH(01011100B)，那么指令：

CLR　A

执行后，累加器的值变为00H(00000000B)。

指令长度(字节)：1

执行周期：1

二进制编码：

1 1 1 0	0 1 0 0

操作：CLR
(A)← **0**

CLR bit

功能：清零指定的位

说明：将bit所代表的位清零，没有标志位会受到影响。CLR可用于进位标志C或者所有可直接寻址的位。

举例：假设端口1的数据为5DH(01011101B)，那么指令

CLR　P1.2

执行后，Pl端口被设置为59H(01011001B)。

STC MCU Limited

CLR C

指令长度(字节)：1

执行周期：1

二进制编码：

1 1 0 0	0 0 1 1

操作：CLR
(C) ← **0**

CLR bit

指令长度(字节)：2

执行周期：1

二进制编码：

1 1 0 0	0 0 1 0	bit address

操作：CLR
(bit) ← **0**

CPL A

功能：累加器*A*求反

说明：将累加器*A*的每一位都取反，即原来为1的位变为0，原来为0的位变为1。该指令不影响标志位。

举例：设累加器*A*的内容为5CH(01011100B)，那么指令

```
CPL   A
```

执行后，累加器的内容变成0A3H（10100011B）。

指令长度(字节)：1

执行周期：1

二进制编码：

1 1 1 1	0 1 0 0

操作：CPL

$(A) \leftarrow \overline{(A)}$

CPL bit

功能：将bit所表示的位求反

说明：将bit变量所代表的位取反，即原来位为1的变为0，原来为0的变为1。没有标志位会受到影响。CPL可用于进位标志C或者所有可直接寻址的位。

注意：如果该指令被用来修改输出端口的状态，那么bit所代表的数据是端口锁存器中的数据，而不是从引脚上输入的当前状态。

举例：设*P*1端口的数据为5BH(01011011B)，那么指令

```
CPL   P1.1

CPL   P1.2
```

执行完后，*P*l端口被设置为5DH(01011101B)。

CPL C

指令长度(字节)：1

执行周期：1

二进制编码：

1 0 1 1	0 0 1 1

操作：CPL

$(C) \leftarrow \overline{(C)}$

CPL bit

指令长度(字节)：2

执行周期：1

二进制编码：

1 0 1 1	0 0 1 0	bit address

操作：CPL

$(bit) \leftarrow \overline{(bit)}$

STC MCU Limited

DA　A

功能： 在加法运算之后，对累加器*A*进行十进制调整

说明： DA指令对累加器*A*中存放的由此前的加法运算产生的8位数据进行调整（ADD或ADDC指令可以用来实现两个压缩BCD码的加法），生成两个4位的数字。

如果累加器的低4位（位3~位0）大于9 (xxxx1010~xxxx 1111），或者加法运算后，辅助进位标志AC为1，那么DA指令将把6加到累加器上，以在低4位生成正确的BCD数字。若加6后，低4位向上有进位，且高4位都为1，进位则会一直向前传递，以致最后进位标志被置1；但在其他情况下进位标志并不会被清零，进位标志会保持原来的值。

如果进位标志为1，或者高4位的值超过9 (1010xxxx~1111xxxx)，那么DA指令将把6加到高4位，在高4位生成正确的BCD数字，但不清除标志位。若高4位有进位输出，则置进位标志为1，否则，不改变进位标志。进位标志的状态指明了原来的两个BCD数据之和是否大于99，因而DA指令使得CPU可以精确地进行十进制的加法运算。注意，OV标志不会受影响。

DA指令的以上操作在一个指令周期内完成。实际上，根据累加器*A*和机器状态字PSW中的不同内容，DA把00H、06H、60H、66H加到累加器*A*上，从而实现十进制转换。

注意：如果前面没有进行加法运算，不能直接用DA指令把累加器*A*中的十六进制数据转换为BCD数，此外，如果先前执行的是减法运算，DA指令也不会有所预期的效果。

举例： 如果累加器中的内容为56H（01010110B），表示十进制数56的BCD码，寄存器3的内容为67H（01100111B），表示十进制数67的BCD码。进位标志为1，则指令

```
ADDC   A,R3
DA     A
```

先执行标准的补码二进制加法，累加器*A*的值变为0BEH，进位标志和辅助进位标志被清零。

接着，DA执行十进制调整，将累加器*A*的内容变为24H（00100100B），表示十进制数24的BCD码，也就是56、67及进位标志之和的后两位数字。DA指令会把进位标志置位1，这表示在进行十进制加法时，发生了溢出。56、67以及1的和为124。

把BCD格式的变量加上0lH或99H，可以实现加1或者减1。假设累加器的初始值为30H(表示十进制数30)，指令序列

```
ADD    A,  #99H
DA     A
```

将把进位C置为1，累加器*A*的数据变为29H，因为30＋99＝129。加法和的低位数据可以看作减法运算的结果，即30－1＝29。

STC MCU Limited

指令长度(字节)： 1

执行周期： 1

二进制编码：

1 1 0 1	0 1 0 0

操作： DA

-contents of Accumulator are BCD

IF [[(A_{3-0}) > 9] V [(AC) = 1]]

THEN(A_{3-0}) ← **(A_{3-0})** + 6

AND

IF [[(A_{7-4}) > 9] V [(C) = 1]]

THEN (A_{7-4}) ← **(A_{7-4})** + 6

DEC byte

功能： 把BYTE所代表的操作数减1

说明： BYTE所代表的变量被减去l。如果原来的值为00H，那么减去1后，变成0FFH。没有标志位会受到影响。该指令支持4种操作数寻址方式：累加器寻址、寄存器寻址、直接寻址和寄存器间接寻址。

注意：当DEC指令用于修改输出端口的状态时，BYTE所代表的数据是从端口输出数据锁存器中获取的，而不是从引脚上读取的输入状态。

举例： 假设寄存器0的内容为7FH（01111111B），内部RAM的7EH和7FH单元的内容分别为00H和40H。则指令

```
DEC   @R0
DEC   R0
DEC   @R0
```

执行后，寄存器0的内容变成7EH，内部RAM的7EH和7FH单元的内容分别变为0FFH和3FH。

DEC A

指令长度(字节)： 1

执行周期： 1

二进制编码：

0 0 0 1	0 1 0 0

操作： DEC

(A)←**(A)** −1

DEC Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

0 0 0 1	1 r r r

操作： DEC

(Rn)←**(Rn) - 1**

STC MCU Limited

DEC　direct

指令长度(字节)：2

执行周期：1

二进制编码：

0 0 0 1	0 1 0 1	direct address

操作：DEC
(direct)←**(direct)** −1

DEC　@Ri

指令长度(字节)：1

执行周期：1

二进制编码：

0 0 0 1	0 1 1 i

操作：DEC
((Ri))←**((Ri)) - 1**

DIV　AB

功能：除法

说明：　DIV指令把累加器*A*中的8位无符号整数除以寄存器*B*中的8位无符号整数，并将商置于累加器*A*中，余数置于寄存器*B*中。进位标志C和溢出标志OV被清零。

例外：如果寄存器*B*的初始值为00H（即除数为0），那么执行DIV指令后，累加器*A*和寄存器*B*中的值是不确定的，且溢出标志OV将被置位。但在任何情况下，进位标志C都会被清零。

举例：假设累加器的值为251 (0FBH或11111011B)，寄存器*B*的值为18 (12H或00010010B)。则指令

DIV　AB

执行后，累加器的值变成13 (0DH或00001101B)，寄存器*B*的值变成17 (11H或0001000B)，正好符合251＝13×18＋17。进位和溢出标志都被清零。

指令长度(字节)：1

执行周期：4

二进制编码：

1 0 0 0	0 1 0 0

操作：DIV
$(A)_{15\text{-}8}$
$(B)_{7\text{-}0}$ ← **(A)/(B)**

STC MCU Limited

DJNZ <byte>, <rel-addr>

功能： 减1，若非0则跳转

说明： DJNZ指令首先将第1个操作数所代表的变量减1，如果结果不为0，则转移到第2个操作数所指定的地址处去执行。如果第1个操作数的值为00H，则减1后变为0FFH。该指令不影响标志位。跳转目标地址的计算：首先将PC值加2（即指向下一条指令的首字节），然后将第2操作数表示的有符号的相对偏移量加到PC上去即可。byte所代表的操作数可采用寄存器寻址或直接寻址。

注意：如果该指令被用来修改输出引脚上的状态，那么byte所代表的数据是从端口输出数据锁存器中获取的，而不是直接读取引脚。

举例： 假设内部RAM的40H、50H和60H单元分别存放着01H、70H和15H，则指令

```
DJNZ    40H, LABEL_1
DJNZ    50H, LABEL_2
DJNZ    60H, LABEL_3
```

执行之后，程序将跳转到标号LABEL2处执行，且相应的3个RAM单元的内容变成00H、6FH和15H。之所以第1个跳转没被执行，是因为减1后其结果为0，不满足跳转条件。

使用DJNZ指令可以方便地在程序中实现指定次数的循环，此外用一条指令就可以在程序中实现中等长度的时间延迟（2~512个机器周期）。指令序列

```
        MOV     R2,#8
TOOOLE: CPL     P1.7
        DJNZ    R2, TOOGLE
```

将使得P1.7的电平翻转8次，从而在P1.7产生4个脉冲，每个脉冲将持续3个机器周期，其中2个为DJNZ指令的执行时间，1个为CPL指令的执行时间。

STC MCU Limited

DJNZ Rn,rel

指令长度（字节）： 2

执行周期： 2

二进制编码：

1 1 0 1	1 r r r	rel. address

操作： DJNZ

(PC) ← **(PC) + 2**

(Rn) ← **(Rn) – 1**

IF (Rn) > 0 or (Rn) < 0

THEN

(PC) ← **(PC)+ rel**

DJNZ　direct, rel

指令长度(字节)：3

执行周期：2

二进制编码：

1 1 0 1	0 1 0 1	direct address	rel. address

操作：DJNZ

(PC) ← **(PC) + 2**

(direct) ← **(direct) – 1**

IF (direct) > 0 or (direct) < 0

THEN

(PC) ← **(PC) + rel**

INC　<byte>

功能：加1

说明：INC指令将<byte>所代表的数据加1。如果原来的值为FFH，则加1后变为00H，该指令步影响标志位。支持3种寻址模式：寄存器寻址、直接寻址、寄存器间接寻址。

注意：如果该指令被用来修改输出引脚上的状态，那么byte所代表的数据是从端口输出数据锁存器中获取的，而不是直接读的引脚。

举例：假设寄存器0的内容为7EH(0111110B)，内部RAM的7E单元和7F单元分别存放着0FFH和40H，则指令序列

```
INC   @R0
INC   R0
INC   @R0
```

执行完毕后，寄存器0的内容变为7FH，而内部RAM的7EH和7FH单元的内容分别变成00H和41H。

INC　A

指令长度(字节)：1

执行周期：1

二进制编码：

0 0 0 0	0 1 0 0

操作：INC

(A) ← **(A)**+1

INC　Rn

指令长度(字节)：1

执行周期：1

二进制编码：

0 0 0 0	1 r r r

操作：INC

(Rn) ← **(Rn)**+1

STC MCU Limited

INC　direct

指令长度(字节)：2

执行周期：1

二进制编码：

0 0 0 0	0 1 0 1	direct address

操作：INC
(direct)←(direct) + 1

INC　@Ri

指令长度(字节)：1

执行周期：1

二进制编码：

0 0 0 0	0 1 1 i

操作：INC
((Ri))←((Ri)) + 1

INC　DPTR

功能：数据指针加1

说明：　该指令实现将DPTR加1功能。需要注意的是，这是16位的递增指令，低位字节DPL从FFH增加1之后变为00H，同时进位到高位字节DPH。该操作不影响标志位。

该指令是唯一1条16位寄存器递增指令。

举例：假设寄存器DPH和DPL的内容分别为12H和0FEH，则指令序列

```
INC　DPTR
INC　DPTR
INC　DPTR
```

执行完毕后，DPH和DPL变成13H和01H

指令长度(字节)：1

执行周期：2

二进制编码：

1 0 1 0	0 0 1 1

操作：INC
(DPTR) ← (DPTR)+1

STC MCU Limited

JB　bit, rel

功能：若位数据为1则跳转

说明：如果bit代表的位数据为1，则跳转到rel所指定的地址处去执行；否则，继续执行下一条指令。跳转的目标地址按照如下方式计算：先增加PC的值，使其指向下一条指令的首字节地址，然后把rel所代表的有符号的相对偏移量（指令的第3个字节）加到PC上去，新的PC值即为目标地址。该指令只是测试相应的位数据，但不会改变其数值，而且该操作不会影响标志位。

举例：假设端口1的输入数据为11001010B，累加器的值为56H（01010110B）。则指令

```
JB     P1.2, LABEL1
JB     ACC.2, LABEL2
```

将导致程序转到标号LABEL2处去执行

指令长度(字节)：3

执行周期：2

二进制编码：

0 0 1 0	0 0 0 0	bit address	rel. address

操作：

```
JB
(PC) ← (PC)+ 3
IF  (bit) = 1
   THEN
        (PC) ← (PC) + rel
```

JBC　bit, rel

功能：若位数据为1则跳转并将其清零

说明：如果bit代表的位数据为1，则将其清零并跳转到rel所指定的地址处去执行。如果bit代表的位数据为0，则继续执行下一条指令。跳转的目标地址按照如下方式计算：先增加PC的值，使其指向下一条指令的首字节地址，然后把rel所代表的有符号的相对偏移量（指令的第3个字节）加到PC上去，新的PC值即为目标地址，而且该操作不会影响标志位。

注意：如果该指令被用来修改输出引脚上的状态，那么byte所代表的数据是从端口输出数据锁存器中获取的，而不是直接读取引脚。

举例：假设累加器的内容为56H(01010110B)，则指令序列

```
JBC     ACC.3, LABEL1
JBC     ACC.2, LABEL2
```

将导致程序转到标号LABEL2处去执行，且累加器的内容变为52H（01010010B）。

指令长度(字节)：3

执行周期：2

二进制编码：

0 0 0 1	0 0 0 0	bit address	rel. address

操作：

```
JBC
(PC) ← (PC)+ 3
IF  (bit) = 1
   THEN
        (bit) ← 0
        (PC) ← (PC) + rel
```

STC MCU Limited

JC　rel

功能： 若进位标志为1，则跳转

说明： 如果进位标志为1，则程序跳转到rel所代表的地址处去执行；否则，继续执行下面的指令。跳转的目标地址按照如下方式计算：先增加PC的值，使其指向紧接JC指令的下一条指令的首地址，然后把rel所代表的有符号的相对偏移量（指令的第2个字节）加到PC上去，新的PC值即为目标地址。该操作不会影响标志位。

举例： 假设进位标志此时为0，则指令序列

```
JC      LABEL1
CPL     C
JC      LABEL2
```

执行完毕后，进位标志变成1，并导致程序跳转到标号LABEL2处去执行。

指令长度（字节）： 2

执行周期： 2

二进制编码：

0 1 0 0	0 0 0 0	rel. address

操作：

```
JC
(PC) ← (PC)+ 2
IF  (C) = 1
   THEN
        (PC) ← (PC) + rel
```

JMP　@A+DPTR

功能： 间接跳转。

说明： 把累加器*A*中的8位无符号数据和16位的数据指针的值相加，其和作为下一条将要执行的指令的地址，传送给程序计数器PC。执行16位的加法时，低字节DPL的进位会传到高字节DPH。累加器*A*和数据指针DPTR 的内容都不会发生变化。不影响任何标志位。

举例： 假设累加器*A*中的值是偶数（从0到6）。下面的指令序列将使得程序跳转到位于跳转表JMP_TBL 的4条AJMP指令中的某一条去执行：

```
                MOV     DPTR, #JMP_TBL
                JMP     @A+DPTR
JMP-TBL:        AJMP    LABEL0
                AJMP    LABEL1
                AJMP    LABEL2
                AJMP    LABEL3
```

如果开始执行上述指令序列时，累加器*A*中的值为04H，那么程序最终会跳转到标号LABEL2处去执行。

注意：AJMP是一个2字节指令，因而在跳转表中，各个跳转指令的入口地址依次相差2个字节。

指令长度（字节）： 1

执行周期： 2

二进制编码：

0 1 1 1	0 0 1 1

操作：

```
JMP
(PC) ← (A) + (DPTR)
```

STC MCU Limited

JNB　bit, rel

功能：如果bit所代表的位不为1则跳转。

说明：如果bit所表示的位为0，则转移到rel所代表的地址去执行；否则，继续执行下一条指令。跳转的目标地址如此计算：先增加PC的值，使其指向下一条指令的首字节地址，然后把rel所代表的有符号的相对偏移量（指令的第3个字节）加到PC上去，新的PC值即为目标地址。该指令只是测试相应的位数据，但不会改变其数值，而且该操作不会影响标志位。

举例：假设端口1的输入数据为110010108，累加器的值为56H（01010110B）。则指令序列

```
JNB     P1.3, LABEL1
JNB     ACC.3, LABEL2
```

执行后将导致程序转到标号LABEL2处去执行。

指令长度(字节)：3

执行周期：2

二进制编码：

0 0 1 1	0 0 0 0	bit address	rel. address

操作：

```
JNB
(PC) ← (PC)+ 3
IF  (bit) = 0
   THEN    (PC) ← (PC) + rel
```

JNC　rel

功能：若进位标志非1则跳转

说明：如果进位标志为0，则程序跳转到rel所代表的地址处去执行；否则，继续执行下面的指令。跳转的目标地址按照如下方式计算：先增加PC的值加2，使其指向紧接JNC指令的下一条指令的地址，然后把rel所代表的有符号的相对偏移量（指令的第2个字节）加到PC上去，新的PC值即为目标地址。该操作不会影响标志位。

举例：假设进位标志此时为1，则指令序列

```
JNC   LABEL1
CPL   C
JNC   LABEL2
```

执行完毕后，进位标志变成0，并导致程序跳转到标号LABEL2处去执行。

指令长度(字节)：2

执行周期：2

二进制编码：

0 1 0 1	0 0 0 0	rel. address

操作：

```
JNC
(PC) ← (PC)+ 2
IF  (C) = 0
   THEN    (PC) ← (PC) + rel
```

STC MCU Limited

JNZ　rel

功能：如果累加器的内容非0则跳转

说明：如果累加器A的任何一位为1，那么程序跳转到rel所代表的地址处去执行，如果各个位都为0，继续执行下一条指令。跳转的目标地址按照如下方式计算：先把PC的值增加2，然后把rel所代表的有符号的相对偏移量（指令的第2个字节）加到PC上去，新的PC值即为目标地址。操作过程中累加器的值不会发生变化，不会影响标志位。

举例：设累加器的初始值为00H，则指令序列

```
JNZ    LABEL1
INC    A
JNZ    LAEEL2
```

执行完毕后，累加器的内容变成01H，且程序将跳转到标号LABEL2处去执行。

指令长度(字节)：2

执行周期：2

二进制编码：

0 1 1 1	0 0 0 0	rel. address

操作：JNZ

(PC) ← **(PC)+ 2**

IF (A) ≠ 0

　THEN　(PC) ← **(PC) + rel**

STC MCU Limited

JZ　rel

功能：若累加器的内容为0则跳转

说明：如果累加器A的任何一位为0，那么程序跳转到rel所代表的地址处去执行，如果各个位都为0，继续执行下一条指令。跳转的目标地址按照如下方式计算：先把PC的值增加2，然后把rel所代表的有符号的相对偏移量（指令的第2个字节）加到PC上去，新的PC值即为目标地址。操作过程中累加器的值不会发生变化，不会影响标志位。

举例：设累加器的初始值为01H，则指令序列

```
JZ     LABEL1
DEC    A
JZ     LAEEL2
```

执行完毕后，累加器的内容变成00H，且程序将跳转到标号LABEL2处去执行。

指令长度(字节)：2

执行周期：2

二进制编码：

0 1 1 0	0 0 0 0	rel. address

操作：JZ

(PC) ← **(PC)+ 2**

IF (A) = 0

　THEN　(PC) ← **(PC) + rel**

LCALL addr16

功能：长调用

说明：LCALL用于调用addr16所指地址处的子例程。首先将PC的值增加3，使得PC指向紧随LCALL的下一条指令的地址，然后把16位PC的低8位和高8位依次压入栈（低位字节在先），同时把栈指针加2。然后再把LCALL指令的第2字节和第3字节的数据分别装入PC的高位字节DPH和低位字节DPL，程序从新的PC所对应的地址处开始执行。因而子例程可以位于64KB程序存储空间的任何地址处。该操作不影响标志位。

举例：栈指针的初始值为07H，标号SUBRTN被分配的程序存储器地址为1234H。则执行如下位于地址0123H的指令后，

LCALL　SUBRTN

栈指针变成09H，内部RAM的08H和09H单元的内容分别为26H和01H，且PC的当前值为1234H。

指令长度(字节)：3

执行周期：2

二进制编码：

0 0 0 1	0 0 1 0	addr15-addr8	addr7-addr0

操作：LCALL

(PC) ← **(PC) + 3**

(SP) ← **(SP) + 1**

((SP)) ← **(PC$_{7-0}$)**

(SP) ← **(SP) + 1**

((SP)) ← **(PC$_{15-8}$)**

(PC) ← **addr$_{15-0}$**

STC MCU Limited

LJMP addr16

功能：长跳转

说明：LJMP使得CPU无条件跳转到addr16所指的地址处执行程序。把该指令的第2字节和第3字节分别装入程序计数器PC的高位字节DPH和低位字节DPL。程序从新PC值对应的地址处开始执行。该16位目标地址可位于64KB程序存储空间的任何地址处。该操作不影响标志位。

举例：假设标号JMPADR被分配的程序存储器地址为1234H。则位于地址1234H的指令

LJMP　JMPADR

执行完毕后，PC的当前值变为1234H。

指令长度(字节)：3

执行周期：2

二进制编码：

0 0 0 0	0 0 1 0	addr15-addr8	addr7-addr0

操作：LJMP

(PC) ← **addr$_{15-0}$**

MOV <dest-byte> , <src-byte>

功能： 传送字节变量

说明： 将第2操作数代表字节变量的内容复制到第1操作数所代表的存储单元中去。该指令不会改变源操作数，也不会影响其他寄存器和标志位。

MOV指令是迄今为止使用最灵活的指令，源操作数和目的操作数组合起来，寻址方式可达15种。

举例： 假设内部RAM的30H单元的内容为40H，而40H单元的内容为10H。端口1 的数据为11001010B（0CAH）。则指令序列

```
MOV     R0, #30H    ;R0< = 30H
MOV     A, @R0      ;A < = 40H
MOV     R1, A       ;R1 < = 40H
MOV     B, @Rl      ;B < = 10H
MOV     @Rl, Pl     ;RAM (40H) < = 0CAH
MOV     P2, P1      ;P2  #0CAH
```

执行完毕后，寄存器0的内容为30H，累加器和寄存器1的内容都为40H，寄存器B的内容为10H，RAM中40H单元和P2口的内容均为0CAH。

MOV A,Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

1 1 1 0	1 r r r

操作： MOV
(A) ← **(Rn)**

***MOV A,direct**

指令长度(字节)： 2

执行周期： 1

二进制编码：

1 1 1 0	0 1 0 1	direct address

操作： MOV
(A)← **(direct)**

注意：**MOV A, ACC**是无效指令。

MOV A,@Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

1 1 1 0	0 1 1 i

操作： MOV
(A) ← **((Ri))**

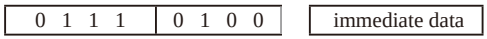
STC MCU Limited

MOV A,#data

指令长度(字节)： 2

执行周期： 1

二进制编码：



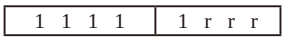
操作： MOV
(A)← #data

MOV Rn, A

指令长度(字节)： 1

执行周期： 1

二进制编码：



操作： MOV
(Rn)←(A)

MOV Rn,direct

指令长度(字节)： 2

执行周期： 2

二进制编码：

操作： MOV
(Rn)←(direct)

MOV Rn,#data

指令长度(字节)： 2

执行周期： 1

二进制编码：

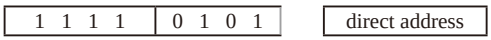
操作： MOV
(Rn) ← #data

MOV direct, A

指令长度(字节)： 2

执行周期： 1

二进制编码：



操作： MOV
(direct) ← (A)

MOV direct, Rn

指令长度(字节)： 2

执行周期： 2

二进制编码：

操作： MOV
(direct) ← (Rn)

STC MCU Limited

MOV　direct, direct

指令长度(字节)：3

执行周期：2

二进制编码：

1 0 0 0	0 1 0 1	dir.addr. (src)

操作：MOV

(direct)← **(direct)**

MOV　direct, @Ri

指令长度(字节)：2

执行周期：2

二进制编码：

1 0 0 0	0 1 1 i	direct addr.

操作：MOV

(direct)←**((Ri))**

MOV　direct,#data

指令长度(字节)：3

执行周期：2

二进制编码：

0 1 1 1	0 1 0 1	direct address

操作：MOV

(direct) ← **#data**

MOV　@Ri, A

指令长度(字节)：1

执行周期：1

二进制编码：

1 1 1 1	0 1 1 i

操作：MOV

((Ri)) ← **(A)**

MOV　@Ri, direct

指令长度(字节)：2

执行周期：2

二进制编码：

1 0 1 0	0 1 1 i	direct addr.

操作：MOV

((Ri)) ← **(direct)**

MOV　@Ri, #data

指令长度(字节)：2

执行周期：1

二进制编码：

0 1 1 1	0 1 1 i	immediate data

操作：MOV

((Ri)) ← **#data**

STC MCU Limited

MOV <dest-bit> , <src-bit>

功能：传送位变量

说明：将<src-bit>代表的布尔变量复制到<dest-bit>所指定的数据单元中去，两个操作数必须有一个是进位标志，而另外一个是可直接寻址的位。本指令不影响其他寄存器和标志位。

举例：假设进位标志C的初值为1，端口P2中的数据是11000101B，端口1的数据被设置为35H(00110101B)。则指令序列

```
MOV     P1.3, C
MOV     C, P3.3
MOV     P1.2, C
```

执行后，进位标志被清零，端口1的数据变为39H（00111001B）。

MOV C,bit

指令长度(字节)：2

执行周期：1

二进制编码：

1	0	1	0	0	0	1	0	bit address

操作：MOV
(C) ← (bit)

MOV bit,C

指令长度(字节)：2

执行周期：2

二进制编码：

1	0	0	1	0	0	1	0	bit address

操作：MOV
(bit)← (C)

MOV DPTR , #data 16

功能：将16位的常数存放到数据指针

说明：该指令将16位常数传递给数据指针DPTR。16位的常数包含在指令的第2字节和第3字节中。其中DPH中存放的是#data16的高字节，而DPL中存放的是#data16的低字节。不影响标志位。

该指令是唯一一条能一次性移动16位数据的指令。

举例：指令:

```
MOV     DPTR, #1234H
```

将立即数1234H装入数据指针寄存器中。DPH的值为12H，DPL的值为34H。

指令长度(字节)：3

执行周期：2

二进制编码：

1	0	0	1	0	0	0	0	immediate data 15-8

操作：MOV
(DPTR) ← #data$_{15\text{-}0}$
DPH DPL ← #data$_{15\text{-}8}$ #data$_{7\text{-}0}$

STC MCU Limited

MOVC　A , @A+ <base-reg>

功能： 把程序存储器中的代码字节数据（常数数据）转送至累加器*A*

说明： MOVC指令将程序存储器中的代码字节或常数字节传送到累加器*A*。被传送的数据字节的地址是由累加器中的无符号8位数据和16位基址寄存器（DPTR或PC）的数值相加产生的。如果以PC为基址寄存器，则在累加器内容加到PC之前，PC需要先增加到指向紧邻MOVC之后的语句的地址；如果是以DPTR为基址寄存器，则没有此问题。在执行16位的加法时，低8位产生的进位会传递给高8位。本指令不影响标志位。

举例： 假设累加器*A*的值处于0~4之间，如下子例程将累加器*A*中的值转换为用DB伪指令（定义字节）定义的4个值之一。

```
REL-PC:  INC     A
         MOVC    A, @A+PC
         RET
         DB      66H
         DB      77H
         DB      88H
         DB      99H
```

如果在调用该子例程之前累加器的值为01H，执行完该子例程后，累加器的值变为77H。MOVC指令之前的INC A指令是为了在查表时越过RET而设置的。如果MOVC和表格之间被多个代码字节所隔开，那么为了正确地读取表格，必须将相应的字节数预先加到累加器*A*上。

STC MCU Limited

MOVC　A,@A+DPTR

指令长度(字节)： 1

执行周期： 2

二进制编码：

1	0	0	1	0	0	1	1

操作： MOVC

(A) ← **((A)+(DPTR))**

MOVC　A,@A+PC

指令长度(字节)： 1

执行周期： 2

二进制编码：

1	0	0	0	0	0	1	1

操作： MOVC

(PC) ← **(PC)**+1

(A) ← **((A)+(PC))**

MOVX　<dest-byte> , <src-byte>

功能： 外部传送

说明： MOVX指令用于在累加器和外部数据存储器之间传递数据。因此在传送指令MOV后附加了X。MOVX又分为两种类型，它们之间的区别在于访问外部数据RAM的间接地址是8位的还是16位的。

对于第1种类型，当前工作寄存器组的R0和R1提供8位地址到复用端口P0。对于外部I/O扩展译码或者较小的RAM阵列，8位的地址已经够用。若要访问较大的RAM阵列，可在端口引脚上输出高位的地址信号。此时可在MOVX指令之前添加输出指令，对这些端口引脚施加控制。

对于第2种类型，通过数据指针DPTR产生16位的地址。当P2端口的输出缓冲器发送DPH的内容时，P2的特殊功能寄存器保持原来的数据。在访问规模较大的数据阵列时，这种方式更为有效和快捷，因为不需要额外指令来配置输出端口。

在某些情况下，可以混合使用两种类型的MOVX指令。在访问大容量的RAM空间时，既可以用数据指针DP在P2端口上输出地址的高位字节，也可以先用某条指令，把地址的高位字节从P2端口上输出，再使用通过R0或Rl间址寻址的MOVX指令。

举例： 假设有一个分时复用地址/数据线的外部RAM存储器，容量为256B（如：Inte1的8155 RAM / I/O / TIMER），该存储器被连接到8051的端口P0上，端口P3被用于提供外部RAM所需的控制信号。端口Pl和P2用作通用输入/输出端口。R0和Rl中的数据分别为12H和34H，外部RAM的34H单元存储的数据为56H，则下面的指令序列：

```
MOVX    A, @R1
MOVX    @R0, A
```

将数据56H复制到累加器A以及外部RAM的12H单元中。

STC MCU Limited

MOVX　A,@Ri

指令长度（字节）： 1

执行周期： 2

二进制编码：

1	1	1	0	0	0	1	i

操作： MOVX
(A) ← **((Ri))**

MOVX　A,@DPTR

指令长度（字节）： 1

执行周期： 2

二进制编码：

1	1	1	0	0	0	0	0

操作： MOVX
(A) ← **((DPTR))**

MOVX　@Ri, A

指令长度(字节)：1

执行周期：2

二进制编码：

1 1 1 1	0 0 1 i

操作：MOVX
((Ri))← **(A)**

MOVX　@DPTR, A

指令长度(字节)：1

执行周期：2

二进制编码：

1 1 1 1	0 0 0 0

操作：MOVX
(DPTR)←**(A)**

STC MCU Limited

MUL　AB

功能：乘法

说明：该指令可用于实现累加器和寄存器B中的无符号8位整数的乘法。所产生的16位乘积的低8位存放在累加器中，而高8位存放在寄存器B中。若乘积大于255(0FFH)，则置位溢出标志；否则清零标志位。在执行该指令时，进位标志总是被清零。

举例：假设累加器A的初始值为80(50H)，寄存器B的初始值为160 (0A0H)，则指令：

MUL　AB

求得乘积12 800 (3200H)，所以寄存器B的值变成32H (00110010B)，累加器被清零，溢出标志被置位，进位标志被清零。

指令长度(字节)：1

执行周期：4

二进制编码：

1 0 1 0	0 1 0 0

操作：MUL
$(A)_{7\text{-}0}$ ← (A)×(B)
$(B)_{15\text{-}8}$

NOP

功能： 空操作

说明： 执行本指令后，将继续执行随后的指令。除了PC外，其他寄存器和标志位都不会有变化。

举例： 假设期望在端口P2的第7号引脚上输出一个长时间的低电平脉冲，该脉冲持续5个机器周期（精确）。若是仅使用SETB和CLR指令序列，生成的脉冲只能持续1个机器周期。因而需要设法增加4个额外的机器周期。可以按照如下方式来实现所要求的功能（假设中断没有被启用）：

```
CLR     P2.7
NOP
NOP
NOP
NOP
SETB    P2.7
```

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	0	0	0	0	0

操作： NOP
(PC) ← (PC)+1

ORL <dest-byte> , <src-byte>

功能： 两个字节变量的逻辑或运算

说明： ORL指令将由<dest-byte>和<src_byte>所指定的两个字节变量进行逐位逻辑或运算，结果存放在<dest-byte>所代表的数据单元中。该操作不影响标志位。

两个操作数组合起来，支持6种寻址方式。当目的操作数是累加器*A*时，源操作数可以采用寄存器寻址、直接寻址、寄存器间接寻址或者立即寻址。当目的操作数采用直接寻址方式时，源操作数可以是累加器或立即数。

注意：如果该指令被用来修改输出引脚上的状态，那么<dest-byte>所代表的数据是从端口输出数据锁存器中获取的数据，而不是从引脚上读取的数据。

举例： 假设累加器*A*中数据为0C3H (11000011B)，寄存器*R*0中的数据为55H(01010101)，则指令：

```
ORL     A, R0
```

执行后，累加器的内容变成0D7H(11010111B)。当目的操作数是直接寻址数据字节时，ORL指令可用来把任何RAM单元或者硬件寄存器中的各个位设置为1。究竟哪些位会被置1由屏蔽字节决定，屏蔽字节既可以是包含在指令中的常数，也可以是累加器*A*在运行过程中实时计算出的数值。执行指令：

```
ORL     P1, #00110010B
```

之后，把1口的第5、4、1位置1。

STC MCU Limited

ORL A,Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	1	0	0	1	r	r	r

操作： ORL

(A) ← **(A)**∨(Rn)

ORL A,direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	0	0	0	1	0	1	direct address

操作： ORL

(A)← **(A)**∨(direct)

ORL A,@Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	1	0	0	0	1	1	i

操作： ORL

(A)← **(A)**∨((Ri))

ORL A,#data

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	0	0	0	1	0	0	immediate data

操作： ORL

(A)← **(A)**∨ #data

ORL direct, A

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	0	0	0	0	1	0	direct address

操作： ORL

(direct)← **(direct)**∨(A)

ORL direct, #data

指令长度(字节)： 3

执行周期： 2

二进制编码：

0	1	0	0	0	0	1	1	direct address	immediate data

操作： ORL

(direct) ← **(direct)**∨#data

STC MCU Limited

ORL　C, <src-bit>

功能：位变量的逻辑或运算

说明：如果<src-bit>所表示的位变量为1，则置位进位标志；否则，保持进位标志的当前状态不变。在汇编语言中，位于源操作数之前的“/”表示将源操作数取反后使用，但源操作数本身不发生变化。在执行本指令时，不影响其他标志位。

举例：当执行如下指令序列时，当且仅当P1.0＝1或ACC.7＝1或OV＝0时，置位进位标志C:

```
MOV     C, P1.0         ;LOAD CARRY WITH INPUT PIN P10
ORL     C, ACC.7        ;OR CARRY WITH THE ACC.BIT 7
ORL     C, /OV          ;OR CARRY WITH THE INVERSE OF OV
```

ORL　C, bit

指令长度(字节)：2

执行周期：2

二进制编码：

0 1 1 1	0 0 1 0	bit address

操作：ORL

(C) ← (C)∨(bit)

ORL　C, /bit

指令长度(字节)：2

执行周期：2

二进制编码：

1 0 1 0	0 0 0 0	bit address

操作：ORL

(C) ← (C)∨$\overline{(bit)}$

POP　direct

功能：出栈

说明：读取栈指针所指定的内部RAM单元的内容，栈指针减1。然后，将读到的内容传送到由direct所指示的存储单元（直接寻址方式）中去。该操作不影响标志位。

举例：　设栈指针的初值为32H，内部RAM的30H~32H单元的数据分别为20H、23H和01H。则执行指令：

```
POP     DPH
POP     DPL
```

之后，栈指针的值变成30H，数据指针变为0123H。此时指令

```
POP     SP
```

将把栈指针变为20H 。

注意：在这种特殊情况下，在写入出栈数据（20H）之前，栈指针先减小到2FH，然后再随着20H的写入，变成20H。

指令长度(字节)：2

执行周期：2

二进制编码：

1 1 0 1	0 0 0 0	direct address

操作：POP

(direct) ← ((SP))

(SP) ← (SP) - 1

STC MCU Limited

PUSH direct

功能：压栈

说明：栈指针首先加1，然后将direct所表示的变量内容复制到由栈指针指定的内部RAM存储单元中去。该操作不影响标志位。

举例：设在进入中断服务程序时栈指针的值为09H，数据指针DPTR的值为0123H。则执行如下指令序列

```
PUSH   DPL
PUSH   DPH
```

之后，栈指针变为0BH，并把数据23H和01H分别存入内部RAM的0AH和0BH存储单元之中。

指令长度(字节)：2

执行周期：2

二进制编码：

1 1 0 0	0 0 0 0	direct address

操作：PUSH

(SP) ← **(SP) + 1**

((SP)) ← **(direct)**

RET

功能：从子例程返回

说明：执行RET指令时，首先将PC值的高位字节和低位字节从栈中弹出，栈指针减2。然后，程序从形成的PC值所对应的地址处开始执行，一般情况下，该指令和ACALL或LCALL配合使用。改指令的执行不影响标志位。

举例：设栈指针的初值为0BH，内部RAM的0AH和0BH存储单元中的数据分别为23H和01H。则指令:

```
RET
```

执行后，栈指针变为09H。程序将从0123H地址处继续执行。

指令长度(字节)：1

执行周期：2

二进制编码：

0 0 1 0	0 0 1 0

操作：RET

(PC_{15-8}) ← **((SP))**

(SP) ← **(SP) -1**

(PC_{7-0}) ← **((SP))**

(SP) ← **(SP) -1**

STC MCU Limited

RETI

功能：中断返回

说明：执行该指令时，首先从栈中弹出PC值的高位和低位字节，然后恢复中断启用，准备接受同优先级的其他中断，栈指针减2。其他寄存器不受影响。但程序状态字PSW不会自动恢复到中断前的状态。程序将继续从新产生的PC值所对应的地址处开始执行，一般情况下是此次中断入口的下一条指令。在执行RETI指令时，如果有一个优先级较低的或同优先级的其他中断在等待处理，那么在处理这些等待中的中断之前需要执行1条指令。

举例：设栈指针的初值为0BH，结束在地址0123H处的指令执行结束期间产生中断，内部RAM的0AH和0BH单元的内容分别为23H和01H。则指令：

RETI

执行完毕后，栈指针变成09H，中断返回后程序继续从0123H地址开始执行。

指令长度(字节)：1

执行周期：2

二进制编码：

0	0	1	1	0	0	1	0

操作：RETI

$(PC_{15-8}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

STC MCU Limited

RL A

功能：将累加器*A*中的数据位循环左移

说明：将累加器中的8位数据均左移1位，其中位7移动到位0。该指令的执行不影响标志位。

举例：设累加器的内容为0C5H（11000101B），则指令

RL　A

执行后，累加器的内容变成8BH（10001011B），且标志位不受影响。

指令长度(字节)：1

执行周期：1

二进制编码：

0	0	1	0	0	0	1	1

操作：RL

$(An+1) \leftarrow (An)$　$n = 0\text{-}6$

$(A0) \leftarrow (A7)$

RLC A

功能： 带进位循环左移

说明： 累加器的8位数据和进位标志一起循环左移1位。其中位7移入进位标志，进位标志的初始状态值移到位0。该指令不影响其他标志位。

举例： 假设累加器*A*的值为0C5H(11000101B)，则指令

```
RLC    A
```

执行后，将把累加器*A*的数据变为8BH(10001011B)，进位标志被置位。

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	1	1	0	0	1	1

操作： RLC

(An+1) ← (**A**n)　n = 0-6

(A0) ← (**C**)

(C) ← (**A**7)

RR A

功能： 将累加器的数据位循环右移

说明： 将累加器的8个数据位均右移1位，位0将被移到位7，即循环右移，该指令不影响标志位。

举例： 设累加器的内容为0C5H（11000101B），则指令

```
RR     A
```

执行后累加器的内容变成0E2H（11100010B），标志位不受影响。

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	0	0	0	1	1

操作： RR

(An) ← (**A**n+1)　n = 0 - 6

(A7) ← (**A**0)

STC MCU Limited

RRC A

功能：带进位循环右移

说明：累加器的8位数据和进位标志一起循环右移1位。其中位0移入进位标志，进位标志的初始状态值移到位7。该指令不影响其他标志位。

举例：假设累加器的值为0C5H(11000101B)，进位标志为0，则指令

```
RRC     A
```

执行后，将把累加器的数据变为62H(01100010B)，进位标志被置位。

指令长度(字节)：1

执行周期：1

二进制编码：

0	0	0	1	0	0	1	1

操作：RRC

$(A_{n+1}) \leftarrow (A_n)$　n = 0-6

$(A_7) \leftarrow (C)$

$(C) \leftarrow (A_0)$

SETB <bit>

功能：置位

说明：SETB指令可将相应的位置1，其操作对象可以是进位标志或其他可直接寻址的位。该指令不影响其他标志位。

举例：设进位标志被清零，端口1的输出状态为34H(00110100B)，则指令

```
SETB    C
SETB    P1.0
```

执行后，进位标志变为1，端口1的输出状态变成35H(00110101B)。

SETB C

指令长度(字节)：1

执行周期：1

二进制编码：

1	1	0	1	0	0	1	1

操作：SETB

$(C) \leftarrow 1$

SETB bit

指令长度(字节)：2

执行周期：1

二进制编码：

1	1	0	1	0	0	1	0	bit address

操作：SETB

$(bit) \leftarrow 1$

STC MCU Limited

SJMP　rel

功能：短跳转

说明：程序无条件跳转到rel所示的地址去执行。目标地址按如下方法计算：首先PC值加2，然后将指令第2字节（即rel）所表示的有符号偏移量加到PC上，得到的新PC值即短跳转的目标地址。所以，跳转的范围是当前指令（即SJMP）地址的前128字节和后127字节。

举例：设标号RELADR对应的指令地址位于程序存储器的0123H地址，则指令：

```
SJMP    RELADR
```

汇编后位于0100H。当执行完该指令后，PC值变成0123H。

注意：在上例中，紧接SJMP的下一条指令的地址是0102H，因此，跳转的偏移量为0123H−0102H＝21H。另外，如果SJMP的偏移量是0FEH，那么构成只有1条指令的无限循环。

指令长度(字节)：2

执行周期：2

二进制编码：

1 0 0 0	0 0 0 0	rel. address

操作：SJMP

(PC) ← (PC)+2

(PC) ← (PC)+rel

SUBB　A, <src-byte>

功能：带借位的减法

说明：SUBB指令从累加器中减去<src-byte>所代表的字节变量的数值及进位标志，减法运算的结果置于累加器中。如果执行减法时第7位需要借位，SUBB将会置位进位标志（表示借位）；否则，清零进位标志。（如果在执行SUBB指令前，进位标志C已经被置位，这意味着在前面进行多精度的减法运算时，产生了借位。因而在执行本条指令时，必须把进位连同源操作数一起从累加器中减去。）如果在进行减法运算的时候，第3位处向上有借位，那么辅助进位标志AC会被置位；如果第6位有借位；而第7位没有，或是第7位有借位，而第6位没有， 则溢出标志OV被置位。

当进行有符号整数减法运算时，若OV置位，则表示在正数减负数的过程中产生了负数；或者，在负数减正数的过程中产生了正数。

源操作数支持的寻址方式：寄存器寻址、直接寻址、寄存器间接寻址和立即数寻址。

举例：设累加器中的数据为0C9H(11001001B)。寄存器*R2*的值为54H(01010100B)，进位标志C被置位。则如下指令:

```
SUBB    A, R2
```

执行后，累加器的数据变为74H(01110100B)，进位标志C和辅助进位标志AC被清零，溢出标志C被置位。

注意：0C9H减去54H应该是75H，但在上面的计算中，由于在SUBB指令执行前，进位标志C已经被置位，因而最终结果还需要减去进位标志，得到74H。因此，如果在进行单精度或者多精度减法运算前，进位标志C的状态未知，那么应改采用CLR　C 指令把进位标志C清零。

STC MCU Limited

SUBB A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	0	0	1	1	r	r	r

操作： SUBB

(A) ← **(A) - (C) - (Rn)**

SUBB A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	0	1	0	1	0	1	direct address

操作： SUBB

(A) ← **(A) - (C) - (direct)**

SUBB A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	0	0	1	0	1	1	i

操作： SUBB

(A) ← **(A) - (C) - ((Ri))**

SUBB A, #data

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	0	1	0	1	0	0	immediate data

操作： SUBB

(A) ← **(A) - (C) - #data**

SWAP A

功能： 交换累加器的高低半字节

说明： SWAP指令把累加器的低4位（位3~位0）和高4位（位7~位4）数据进行交换。实际上SWAP指令也可视为4位的循环指令。该指令不影响标志位。

举例： 设累加器的内容为0C5H（11000101B），则指令

SWAP A

执行后，累加器的内容变成5CH（01011100B）。

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	0	0	1	0	0

操作： SWAP

$(A_{3\text{-}0}) \rightleftarrows (A_{7\text{-}4})$

STC MCU Limited

XCH　A, <byte>

功能： 交换累加器和字节变量的内容

说明： XCH指令将<byte>所指定的字节变量的内容装载到累加器，同时将累加器的旧内容写入<byte>所指定的字节变量。指令中的源操作数和目的操作数允许的寻址方式：寄存器寻址、直接寻址和寄存器间接寻址。

举例： 设R0的内容为地址20H，累加器的值为3FH (00111111B)。内部RAM的20H单元的内容为75H (01110101B)。则指令

```
XCH     A, @R0
```

执行后，内部RAM的20H单元的数据变为3FH (00111111B)，累加器的内容变为75H(01110101B)。

XCH　A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

1 1 0 0	1 r r r

操作： XCH

(A) ⇄ (Rn)

XCH　A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

1 1 0 0	0 1 0 1	direct address

操作： XCH

(A) ⇄ (direct)

XCH　A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

1 1 0 0	0 1 1 i

操作： XCH

(A) ⇄ ((Ri))

STC MCU Limited

XCHD　A, @Ri

功能：交换累加器和@Ri对应单元中的数据的低4位

说明：XCHD指令将累加器内容的低半字节（位0~3，一般是十六进制数或BCD码）和间接寻址的内部RAM单元的数据进行交换，各自的高半字（位7~4）节不受影响。另外，该指令不影响标志位。

举例：设R0保存了地址20H，累加器的内容为36H (00110110B)。内部RAM的20H单元存储的数据为75H (011110101B)。则指令:

```
XCHD      A, @R0
```

执行后，内部RAM 20H单元的内容变成76H (01110110B)，累加器的内容变为35H(00110101B)。

指令长度(字节)：1

执行周期：1

二进制编码：

1	1	0	1	0	1	1	i

操作：XCHD

$(A_{3\text{-}0}) \rightleftarrows (Ri_{3\text{-}0})$

XRL　<dest-byte>, <src-byte>

功能：字节变量的逻辑异或

说明：　XRL指令将<dest-byte>和<src-byte>所代表的字节变量逐位进行逻辑异或运算，结果保存在<dest-byte>所代表的字节变量里。该指令不影响标志位。

两个操作数组合起来共支持6种寻址方式：当目的操作数为累加器时，源操作数可以采用寄存器寻址、直接寻址、寄存器间接寻址和立即数寻址；当目的操作数是可直接寻址的数据时，源操作数可以是累加器或者立即数。

注意：如果该指令被用来修改输出引脚上的状态，那么dest-byte所代表的数据就是从端口输出数据锁存器中获取的数据，而不是从引脚上读取的数据。

举例：　如果累加器和寄存器0的内容分别为0C3H (11000011B)和0AAH(10101010B)，则指令：

```
XRL       A, R0
```

执行后，累加器的内容变成69H (01101001B)。

当目的操作数是可直接寻址字节数据时，该指令可把任何RAM单元或者寄存器中的各个位取反。具体哪些位会被取反，在运行过程当中确定。指令：

```
XRL       P1, #00110001B
```

执行后，P1口的位5、4、0被取反。

STC MCU Limited

XRL　A, Rn

指令长度(字节)：1

执行周期：1

二进制编码：

0 1 1 0	1 r r r

操作：XRL

(A) ← (A) ⊻ (Rn)

XRL　A, direct

指令长度(字节)：2

执行周期：1

二进制编码：

0 1 1 0	0 1 0 1	direct address

操作：XRL

(A) ← (A) ⊻ (direct)

XRL　A, @Ri

指令长度(字节)：1

执行周期：1

二进制编码：

0 1 1 0	0 1 1 i

操作：XRL

(A) ← (A) ⊻ ((Ri))

XRL　A, #data

指令长度(字节)：2

执行周期：1

二进制编码：

0 1 1 0	0 1 0 0	immediate data

操作：XRL

(A) ← (A) ⊻ #data

XRL　direct, A

指令长度(字节)：2

执行周期：1

二进制编码：

0 1 1 0	0 0 1 0	direct address

操作：XRL

(direct) ← (direct) ⊻ (A)

XRL　direct, #data

指令长度(字节)：3

执行周期：2

二进制编码：

0 1 1 0	0 0 1 1	direct address	immediate data

操作：XRL

(direct) ← (direct) ⊻ # data

STC MCU Limited

5.3.2 Instruction Definitions of Traditional 8051 MCU

ACALL addr 11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address.The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by suceesively concatenating the five high-order bits of the incremented PC opcode bits 7-5,and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label “SUBRTN” is at program memory location 0345H. After executingthe instruction,

ACALL SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 1	0 0 1 0	a7 a6 a5 a4	a3 a2 a1 a0

Operation: ACALL
(PC)← **(PC)+ 2**
(SP)←**(SP) + 1**
((SP)) ← **(PC_{7-0})**
(SP)←**(SP) + 1**
((SP))←**(PC_{15-8})**
(PC_{10-0})← **page address**

ADD A,<src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register,direct register-indirect, or immediate.

Example: The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A,R0

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

STC MCU Limited

ADD A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	1 r r r

Operation: ADD
(A)←**(A) + (Rn)**

ADD A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 1	direct address

Operation: ADD
(A)←**(A) + (direct)**

ADD A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	0 1 1 i

Operation: ADD
(A)←**(A) + ((Ri))**

ADD A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 0	immediate data

Operation: ADD
(A)←**(A) + #data**

STC MCU Limited

ADDC A,<src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the Carry flag and the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.
OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.
Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B) with the Carry. The instruction,
ADDC A,R0
will leave 6EH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADDC A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	1 r r r

Operation: ADDC

(A)←(A) + (C) + (Rn)

ADDC A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 1	direct address

Operation: ADDC

(A)←(A) + (C) + (direct)

ADDC A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 1 1 i

Operation: ADDC

(A)←(A) + (C) + ((Ri))

ADDC A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 0	immediate data

Operation: ADDC

(A)←(A) + (C) + #data

AJMP addr 11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: The label “JMPADR” is at program memory location 0123H. The instruction,

AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 0	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: AJMP

(PC)← (PC)+ 2

($PC_{10\text{-}0}$)← page address

STC MCU Limited

ANL <dest-byte> , <src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch not the input pins.

Example: If the Accumulator holds 0C3H(11000011B) and register 0 holds 55H (01010101B) then the instruction,

ANL A,R0

will leave 41H (01000001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The instruction,

ANL Pl, #01110011B

will clear bits 7, 3, and 2 of output port 1.

STC MCU Limited

ANL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	1 r r r

Operation: ANL
(A)←**(A)** ∧ (Rn)

ANL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 1	direct address

Operation: ANL
(A)←**(A)** ∧ (direct)

ANL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	0 1 1 i

Operation: ANL
(A)←**(A)** ∧ ((Ri))

ANL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 0	immediate data

Operation: ANL
(A)←**(A)** ∧ #data

ANL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 0 1 0	direct address

Operation: ANL
(direct)←**(direct)** ∧ (A)

ANL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 0 1	0 0 1 1	direct address	immediate data

Operation: ANL
(direct)←**(direct)** ∧ #data

STC MCU Limited

ANL C , <src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash (" / ") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affceted.* No other flsgs are affected.

Only direct addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```
MOV   C, P1.0          ;LOAD CARRY WITH INPUT PIN STATE
ANL   C, ACC.7         ;AND CARRY WITH ACCUM. BIT.7
ANL   C, /OV           ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C,bit

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 1 0	bit address

Operation: ANL
(C) ← **(C)** ∧ (bit)

ANL C, /bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 1	0 0 0 0	bit address

Operation: ANL

(C)←**(C)** ∧ $\overline{\text{(bit)}}$

CJNE <dest-byte>, <src-byte>, rel

Function: Compare and Jump if Not Equal

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence

```
                CJNE    R7,#60H, NOT-EQ
;               . . .   . . . . . .         ; R7 = 60H.
NOT_EQ:         JC      REQ_LOW             ; IF R7 < 60H.
;               . . .   . . . . .           ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT-EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the instruction,

WAIT: CJNE A,P1,WAIT

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on Pl, the program will loop at this point until the P1 data changes to 34H.)

CJNE A,direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 1	direct address	rel. address

Operation:

(PC) ← **(PC) + 3**
IF (A) < > *(direct)*
THEN
　　(PC) ← **(PC) +** *relative offset*
IF (A) < *(direct)*
THEN
　　(C) ← **1**
ELSE
　　(C) ← **0**

STC MCU Limited

CJNE A,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 1	immediata data	rel. address

Operation:

(PC) ← **(PC) + 3**
IF (A) < > *(data)*
THEN
　　(PC) ← **(PC) +** *relative offset*
IF (A) < *(data)*
THEN
　　(C) ← **1**
ELSE
　　(C) ← **0**

CJNE Rn,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	1 r r r	immediata data	rel. address

Operation:

(PC) ← **(PC) + 3**
IF (Rn) < > *(data)*
THEN
　　(PC) ← **(PC) +** *relative offset*
IF (Rn) < *(data)*
THEN
　　(C) ← **1**
ELSE
　　(C) ← **0**

CJNE @Ri,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 1 i	immediate data	rel. address

Operation:

(PC) ← **(PC) + 3**
IF ((Ri)) < > *(data)*
THEN
　　(PC) ← **(PC) +** *relative offset*
IF ((Ri)) < *(data)*
THEN
　　(C) ← **1**
ELSE
　　(C) ← **0**

STC MCU Limited

CLR A

Function: Clear Accumulator

Description: The Aecunmlator is cleared (all bits set on zero). No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the Accumulator set to 00H (00000000B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	0 1 0 0

Operation: CLR

(A)← 0

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 0 1 1

Operation: CLR

(C) ← 0

CLR bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 0 1 0	bit address

Operation: CLR

(bit) ← 0

STC MCU Limited

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

Example: The Accumulator contains 5CH(01011100B). The instruction,

```
CPL   A
```

will leave the Accumulator set to 0A3H (101000011B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 0 0

Operation: CPL

$(A) \leftarrow \overline{(A)}$

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

Note:When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: Port 1 has previously been written with 5BH(01011011B). The instruction,

```
CPL   P1.1
CPL   P1.2
```

will leave the port set to 5DH(01011101B).

CPL C

Bytes: 1

Cycles: 1

Encoding:

1 0 1 1	0 0 1 1

Operation: CPL

$(C) \leftarrow \overline{(C)}$

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 1	0 0 1 0	bit address

Operation: CPL

$(bit) \leftarrow \overline{(bit)}$

STC MCU Limited

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits.Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set or if the four high-order bits now exceed nine(1010xxxx-111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A cannot simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The Accumulator holds the value 56H(01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67.The carry flag is set. The instruction sequence.

```
ADDC   A,R3
DA     A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56,67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD    A,#99H
DA     A
```

will leave the carry set and 29H in the Accumulator, since 30+99=129. The low-order byte of the sum can be interpreted to mean 30 – 1 = 29.

STC MCU Limited

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 1 0 0

Operation: DA
-contents of Accumulator are BCD
IF [[(A_{3-0}) > 9] V [(AC) = 1]]
THEN(A_{3-0}) ← **(A_{3-0})** + 6
AND
IF [[(A_{7-4}) > 9] V [(C) = 1]]
THEN (A_{7-4}) ← **(A_{7-4})** + 6

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH.
No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.
Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC   @R0
DEC   R0
DEC   @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

STC MCU Limited

DEC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 0 0

Operation: DEC
(A)←**(A)** −1

DEC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	1 r r r

Operation: DEC
(Rn)←**(Rn) - 1**

DEC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 1	0 1 0 1	direct address

Operation: DEC
(direct)←(direct) −1

DEC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 1 i

Operation: DEC
((Ri))←((Ri)) - 1

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The Accumulator contains 251(OFBH or 11111011B) and B contains 18(12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010010B) in B, since 251 = (13×18) + 17. Carry and OV will both be cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 0 0	0 1 0 0

Operation: DIV
$(A)_{15-8}$
$(B)_{7-0}$ ← (A)/(B)

STC MCU Limited

DJNZ <byte>, <rel-addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are afected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ    40H, LABEL_1
DJNZ    50H, LABEL_2
DJNZ    60H, LABEL_3
```

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was not taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction The instruction sequence,

```
            MOV     R2,#8
TOOOLE:     CPL     P1.7
            DJNZ    R2, TOOGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

STC MCU Limited

DJNZ Rn,rel

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	1 r r r	rel. address

Operation: DJNZ

(PC) ← **(PC) + 2**
(Rn) ← **(Rn) – 1**
IF (Rn) > 0 or (Rn) < 0
THEN
(PC) ← **(PC)+ rel**

DJNZ direct, rel

Bytes: 3

Cycles: 2

Encoding:

1 1 0 1	0 1 0 1	direct address	rel. address

Operation: DJNZ
(PC) ← **(PC) + 2**
(direct) ← **(direct) – 1**
IF (direct) > 0 or (direct) < 0
THEN
(PC) ← **(PC) + rel**

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H.No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7EH (011111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC   @R0
INC    R0
INC   @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 0 0

Operation: INC
(A) ← (A)+1

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	1 r r r

Operation: INC
(Rn) ← (Rn)+1

INC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 0	0 1 0 1	direct address

Operation: INC
(direct)←**(direct)** + 1

STC MCU Limited

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 1 i

Operation: INC
((Ri))←((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order-byte (DPH). No flags are affected.
This is the only 16-bit register which can be incremented.

Example: Register DPH and DPL contains 12H and 0FEH,respectively. The instruction sequence,
INC DPTR
INC DPTR
INC DPTR
will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 2

Encoding:

1 0 1 0	0 0 1 1

Operation: INC
(DPTR) ← (DPTR)+1

JB bit, rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified. No flags are affected.*

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,
JB P1.2, LABEL1
JB ACC.2, LABEL2
will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 0	0 0 0 0	bit address	rel. address

Operation: JB
(PC) ← (PC)+ 3
IF (bit) = 1
THEN
(PC) ← (PC) + rel

STC MCU Limited

JBC bit, rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one,branch to the address indicated;otherwise proceed with the next instruction.*The bit wili not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: The Accumulator holds 56H (01010110B). The instruction sequence,

```
JBC     ACC.3, LABEL1
JBC     ACC.2, LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 0 0	bit address	rel. address

Operation:

```
JBC
(PC) ← (PC)+ 3
IF  (bit) = 1
   THEN
        (bit) ← 0
        (PC) ← (PC) + rel
```

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice.No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

```
JC      LABEL1
CPL     C
JC      LABEL2s
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 0	0 0 0 0	rel. address

Operation:

```
JC
(PC) ← (PC)+ 2
IF  (C) = 1
   THEN
        (PC) ← (PC) + rel
```

STC MCU Limited

JMP @A+DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
                MOV     DPTR, #JMP_TBL
                JMP     @A+DPTR
JMP-TBL:        AJMP    LABEL0
                AJMP    LABEL1
                AJMP    LABEL2
                AJMP    LABEL3
```

If the Accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 2

Encoding:

0 1 1 1	0 0 1 1

Operation: JMP
(PC) ← **(A) + (DPTR)**

STC MCU Limited

JNB bit, rel

Function: Jump if Bit is not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

```
JNB     P1.3, LABEL1
JNB     ACC.3, LABEL2
```

will cause program execution to continue at the instruction at label LABEL2

Bytes: 3

Cycles: 2

Encoding:

0 0 1 1	0 0 0 0	bit address	rel. address

Operation: JNB
(PC) ← **(PC)+ 3**
IF (bit) = 0
THEN (PC) ← **(PC) + rel**

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified

Example: The carry flag is set. The instruction sequence,

```
JNC   LABEL1
CPL   C
JNC   LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 1	0 0 0 0	rel. address

Operation:

```
JNC
(PC) ← (PC)+ 2
IF  (C) = 0
    THEN    (PC) ← (PC) + rel
```

JNZ rel

Function: Jump if Accumulator Not Zero

Description: If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally holds 00H. The instruction sequence,

```
JNZ   LABEL1
INC   A
JNZ   LAEEL2
```

will set the Accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 0 0	rel. address

Operation:

```
JNZ
(PC) ← (PC)+ 2
IF  (A) ≠ 0
    THEN    (PC) ← (PC) + rel
```

STC MCU Limited

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally contains 01H. The instruction sequence,
```
JZ    LABEL1
DEC   A
JZ    LAEEL2
```
will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 0	0 0 0 0	rel. address

Operation: JZ
(PC) ← **(PC)+ 2**
IF (A) = 0
THEN (PC) ← **(PC) + rel**

LCALL addr16

Function: Long call

Description: LCALL calls a subroutine loated at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: Initially the Stack Pointer equals 07H. The label “SUT2N” is assigned to program memory location 1234H. After executing the instruction,

```
LCALL   SUT2N
```

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LCALL
(PC) ← **(PC) + 3**
(SP) ← **(SP) + 1**
((SP)) ← **(PC_{7-0})**
(SP) ← **(SP) + 1**
((SP)) ← **(PC_{15-8})**
(PC) ← **$addr_{15-0}$**

STC MCU Limited

LJMP addr16

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

Example: The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

LJMP JMPADR

at location 0123H will load the program counter with 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 0	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LJMP

(PC) ← $addr_{15-0}$

MOV <dest-byte> , <src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV     R0, #30H    ;R0< = 30H
MOV     A, @R0      ;A < = 40H
MOV     R1, A       ;R1 < = 40H
MOV     B, @Rl      ;B < = 10H
MOV     @Rl, Pl     ;RAM (40H) < = 0CAH
MOV     P2, P1      ;P2  #0CAH
```

leaves the value 30H in register 0,40H in both the Accumulator and register 1,10H in register B, and 0CAH(11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	1 r r r

Operation: MOV

(A) ← **(Rn)**

STC MCU Limited

***MOV A,direct**

Bytes: 2

Cycles: 1

Encoding:

1 1 1 0	0 1 0 1	direct address

Operation: MOV
(A)← **(direct)**

***MOV A, ACC is not a valid instruction**

MOV A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	0 1 1 i

Operation: MOV
(A) ← **((Ri))**

MOV A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 0 0	immediate data

Operation: MOV
(A)← **#data**

MOV Rn, A

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	1 r r r

Operation: MOV
(Rn)←**(A)**

MOV Rn,direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	1 r r r	direct addr.

Operation: MOV
(Rn)←**(direct)**

MOV Rn,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	1 r r r	immediate data

Operation: MOV
(Rn) ← **#data**

STC MCU Limited

MOV direct, A

Bytes: 2

Cycles: 1

Encoding:

1 1 1 1	0 1 0 1	direct address

Operation: MOV
(direct) ← **(A)**

MOV direct, Rn

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	1 r r r	direct address

Operation: MOV
(direct) ← **(Rn)**

MOV direct, direct

Bytes: 3

Cycles: 2

Encoding:

1 0 0 0	0 1 0 1	dir.addr. (src)

Operation: MOV
(direct)← **(direct)**

MOV direct, @Ri

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 1 1 i	direct addr.

Operation: MOV
(direct)←**((Ri))**

MOV direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 1 1	0 1 0 1	direct address

Operation: MOV
(direct) ← **#data**

MOV @Ri, A

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 1 i

Operation: MOV
((Ri)) ← **(A)**

STC MCU Limited

MOV @Ri, direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 1 1 i	direct addr.

Operation: MOV
((Ri)) ← (direct)

MOV @Ri, #data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 1 i	immediate data

Operation: MOV
((Ri)) ← #data

MOV <dest-bit> , <src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

```
MOV     P1.3, C
MOV     C, P3.3
MOV     P1.2, C
```

will leave the carry cleared and change Port 1 to 39H (00111001B).

MOV C,bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 0	0 0 1 0	bit address

Operation: MOV
(C) ← (bit)

MOV bit,C

Bytes: 2

Cycles: 2

Encoding:

1 0 0 1	0 0 1 0	bit address

Operation: MOV
(bit)← (C)

STC MCU Limited

MOV DPTR , #data 16

Function: Load Data Pointer with a 16-bit constant

Description: The Data Pointer is loaded with the 16-bit constant indicated.The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.
This is the only instruction which moves 16 bits of data at once.

Example: The instruction,
MOV DPTR, #1234H
will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3

Cycles: 2

Encoding:

1 0 0 1	0 0 0 0	immediate data 15-8

Operation: MOV
(DPTR) ← #data$_{15\text{-}0}$
DPH DPL ← #data$_{15\text{-}8}$ #data$_{7\text{-}0}$

MOVC A , @A+ <base-reg>

Function: Move Code byte

Description: The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit. Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defimed by the DB (define byte) directive.

```
REL-PC:  INC    A
         MOVC   A, @A+PC
         RET
         DB     66H
         DB     77H
         DB     88H
         DB     99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the Accumulator. The INC A before the MOVC instruction is needed to “get around” the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

MOVC A,@A+DPTR

Bytes: 1

Cycles: 2

Encoding:

1 0 0 1	0 0 1 1

Operation: MOVC
(A) ← ((A)+(DPTR))

STC MCU Limited

MOVC A,@A+PC

Bytes: 1

Cycles: 2

Encoding:

1 0 0 0	0 0 1 1

Operation: MOVC
(PC) ← (PC)+1
(A) ← ((A)+(PC))

MOVX <dest-byte> , <src-byte>

Function: Move External

Description: The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX    A, @R1
MOVX    @R0, A
```

copies the value 56H into both the Accumulator and external RAM location 12H.

MOVX A,@Ri

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 1 i

Operation: MOVX
(A) ← ((Ri))

STC MCU Limited

MOVX A,@DPTR

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 0 0

Operation: MOVX
(A) ← **((DPTR))**

MOVX @Ri, A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 1 i

Operation: MOVX
((Ri))← **(A)**

MOVX @DPTR, A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 0 0

Operation: MOVX
(DPTR)←**(A)**

STC MCU Limited

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared

Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 1 0	0 1 0 0

Operation: MUL
$(A)_{7\text{-}0}$ ← (A)×(B)
$(B)_{15\text{-}8}$

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence.

```
CLR     P2.7
NOP
NOP
NOP
NOP
SETB    P2.7
```

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	0	0	0

Operation: NOP
(PC) ← (PC)+1

ORL <dest-byte> , <src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

```
ORL     A, R0
```

will leave the Accumulator holding the value 0D7H (11010111B).
When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time.The instruction,

```
ORL     P1, #00110010B
```

will set bits 5,4, and 1of output Port 1.

STC MCU Limited

ORL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	0	0	1	r	r	r

Operation: ORL
(A) ← **(A)**∨(Rn)

ORL A,direct

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	1	0	1	direct address

Operation: ORL
(A)← **(A)**∨(direct)

ORL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	0	0	0	1	1	i

Operation: ORL
(A)← **(A)**∨((Ri))

ORL A,#data

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	1	0	0	immediate data

Operation: ORL
(A)← **(A)**∨ #data

ORL direct, A

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	0	1	0	direct address

Operation: ORL
(direct)← **(direct)**∨(A)

ORL direct, #data

Bytes: 3

Cycles: 2

Encoding:

0	1	0	0	0	0	1	1	direct address	immediate data

Operation: ORL
(direct) ← **(direct)**∨#data

STC MCU Limited

ORL C, <src-bit>

Function: Logical-OR for bit variables

Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash (" / ") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

```
MOV     C, P1.0         ;LOAD CARRY WITH INPUT PIN P1.0
ORL     C, ACC.7        ;OR CARRY WITH THE ACC.BIT 7
ORL     C, /OV          ;OR CARRY WITH THE INVERSE OF OV
```

ORL C, bit

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 1 0	bit address

Operation: ORL

(C) ← (C)∨(bit)

ORL C, /bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 0 0 0	bit address

Operation: ORL

(C) ← (C)∨$\overline{(bit)}$

POP direct

Function: Pop from stack

Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

```
POP     DPH
POP     DPL
```

will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,

```
POP     SP
```

will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	0 0 0 0	direct address

Operation: POP

(direct) ← ((SP))

(SP) ← (SP) - 1

STC MCU Limited

PUSH direct

Function: Push onto stack

Description: The Stack Pointer is incremented by one. The contents of the indicated variableis then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

Example: On entering interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

```
PUSH    DPL
PUSH    DPH
```

will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2

Cycles: 2

Encoding:

1 1 0 0	0 0 0 0	direct address

Operation: PUSH
(SP) ← **(SP) + 1**
((SP)) ← **(direct)**

RET

Function: Return from subroutine

Description: RET pops the high-and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Example: The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

```
RET
```

will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0 0 1 0	0 0 1 0

Operation: RET
(PC_{15-8}) ← **((SP))**
(SP) ← **(SP) -1**
(PC_{7-0}) ← **((SP))**
(SP) ← **(SP) -1**

STC MCU Limited

RETI

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Example: The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0	0	1	1	0	0	1	0

Operation: RETI
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

STC MCU Limited

RL A

Function: Rotate Accumulator Left

Description: The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

RL A

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	0	0	1	1

Operation: RL
$(An+1) \leftarrow (An)$　n = 0-6
$(A0) \leftarrow (A7)$

RLC A

Function: Rotate Accumulator Left through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,
RLC A
leaves the Accumulator holding the value 8BH (10001011B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 0 1 1

Operation: RLC
(An+1) ← (An)　n = 0-6
(A0) ← (C)
(C) ← (A7)

RR A

Function: Rotate Accumulator Right

Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,
RR A
leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 0 1 1

Operation: RR
(An) ← (An+1)　n = 0 - 6
(A7) ← (A0)

RRC A

Function: Rotate Accumulator Right through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position.No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,
RRC A
leaves the Accumulator holding the value 62H (01100010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 0 1 1

Operation: RRC
(An+1) ← (An)　n = 0-6
(A7) ← (C)
(C) ← (A0)

STC MCU Limited

SETB <bit>

Function: Set bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected

Example: The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions,
SETB C
SETB P1.0
will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 0 1 1

Operation: SETB
(C) ← **1**

SETB bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 1	0 0 1 0	bit address

Operation: SETB
(bit) ← **1**

SJMP rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128bytes preceding this instruction to 127 bytes following it.

Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,
SJMP RELADR
will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.
(*Note:* Under the above conditions the instruction following SJMP will be at 102H.Therefore, the displacement byte of the instruction will be the relative offset (0123H - 0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be an one-instruction infinite loop).

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 0 0	rel. address

Operation: SJMP
(PC) ← **(PC)+2**
(PC) ← **(PC)+rel**

STC MCU Limited

SUBB A, <src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow)flag if a borrow is needed for bit 7, and clears C otherwise.(If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand).AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A, R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

STC MCU Limited

SUBB A, Rn

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	1 r r r

Operation: SUBB
(A) ← (A) - (C) - (Rn)

SUBB A, direct

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 1	direct address

Operation: SUBB
(A) ← (A) - (C) - (direct)

SUBB A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	0 1 1 i

Operation: SUBB
(A) ← (A) - (C) - ((Ri))

SUBB A, #data

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 0	immediate data

Operation: SUBB

(A) ← (A) - (C) - #data

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the Accumulator holding the value 5CH (01011100B).

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 0 0

Operation: SWAP

$(A_{3-0}) \rightleftarrows (A_{7-4})$

XCH A, <byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A, @R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A, Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	1 r r r

Operation: XCH

(A) $\rightleftarrows$ (Rn)

XCH A, direct

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 1 0 1	direct address

Operation: XCH

(A) $\rightleftarrows$ (direct)

STC MCU Limited

XCH A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 1 i

Operation: XCH

(A) ⇄ ((Ri))

XCHD A, @Ri

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A, @R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 1 1 i

Operation: XCHD

(A_{3-0}) ⇄ (Ri_{3-0})

XRL <dest-byte>, <src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations.When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address,the source can be the Accumulator or immediate data.

(*Note*: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.)

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A, R0

will leave the Accumulator holding the vatue 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinnation of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

XRL P1, #00110001B

will complement bits 5,4 and 0 of outpue Port 1.

STC MCU Limited

XRL　A, Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 1 0	1 r r r

Operation: XRL
(A) ← (A) ⅄ (Rn)

XRL　A, direct

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 1 0 1	direct address

Operation: XRL
(A) ← (A) ⅄ (direct)

XRL　A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 1 0	0 1 1 i

Operation: XRL
(A) ← (A) ⅄ ((Ri))

XRL　A, #data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 1 0 0	immediate data

Operation: XRL
(A) ← (A) ⅄ #data

XRL　direct, A

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 0 1 0	direct address

Operation: XRL
(direct) ← (direct) ⅄ (A)

XRL　direct, #data

Bytes: 3

Cycles: 2

Encoding:

0 1 1 0	0 0 1 1	direct address	immediate data

Operation: XRL
(direct) ← (direct) ⅄ # data

STC MCU Limited

第6章 中断系统

中断系统是为使CPU具有对外界**紧急事件的实时处理能力而设置的。**

当中央处理机CPU正在处理某件事的时候外界发生了紧急事件请求，要求CPU暂停当前的工作，转而去处理这个紧急事件，**处理完以后，再回到原来被中断的地方，继续原来的工作，**这样的过程称为中断。实现这种功能的部件称为中断系统，请示CPU中断的请求源称为中断源。微型机的中断系统一般允许多个中断源，当几个中断源同时向CPU请求中断，要求为它服务的时候，这就存在CPU优先响应哪一个中断源请求的问题。通常根据中断源的轻重缓急排队，优先处理最紧急事件的中断请求源，即规定每一个中断源有一个优先级别。CPU总是先响应**优先级别最高的中断请求。**

当CPU正在处理一个中断源请求的时候（执行相应的**中断服务程序），发生了另外一个优**先级比它还高的中断源请求。如果CPU能够暂停对原来中断源的**服务程序，转而去处理优先级**更高的中断请求源，处理完以后，再回到原低级中断**服务程序，这样的过程称为中断嵌套。这**样的中断系统称为多级中断系统，没有中断嵌套功能的中断系统称为单级中断系统。

STC15W4K60S4系列单片机提供了19个中断请求源，它们分别是：外部中断0(INT0)、定时器0中断、外部中断1(INT1)、定时器1中断、串口1中断、A/D转换中断、低压检测(LVD)中断、CCP/PWM/PCA中断、串口2中断、SPI中断、外部中断2(INT2)、外部中断3(INT3)、定时器T2中断、外部中断4(INT4)、串口3中断、串口4中断、定时器3中断、定时器4中断以及比较器中断。除外部中断2(INT2)、外部中断3(INT3)、定时器T2中断、外部中断4(INT4)、串口3中断、串口4中断、定时器3中断、定时器4中断及比较器中断固定是最低优先级中断外，其它的中断都具有2个中断优先级，可实现2级中断服务程序嵌套。用户可以用关总中断允许位(EA/IE.7)或相应中断的允许位屏蔽相应的中断请求，也可以用打开相应的中断允许位来使CPU响应相应的中断申请；每一个中断源可以用软件独立地控制为开中断或关中断状态；部分中断的优先级别均可用软件设置。高优先级的中断请求可以打断低优先级的中断，反之，低优先级的中断请求不可以打断高优先级的中断。当两个相同优先级的中断同时产生时，将由查询次序来决定系统先响应哪个中断。

6.1 STC15系列单片机的中断请求源

STC15全系列的中断请求源的类型如下表所示。

单片机型号 / 中断源类型	STC15F101W系列	STC15F408AD系列	STC15W201S系列	STC15W408AS系列	STC15W408S系列	STC15W1K16S系列	STC15F2K60S2系列	STC15W4K60S4系列
外部中断0 (INT0)	√	√	√	√	√	√	√	√
定时器0中断	√	√	√	√	√	√	√	√
外部中断1 (INT1)	√	√	√	√	√	√	√	√
定时器1中断					√	√	√	√
串口1中断		√	√	√	√	√	√	√
A/D转换中断		√		√			√	√
低压检测(LVD)中断	√	√	√	√	√	√	√	√
CCP/PWM/PCA中断		√		√			√	√
串口2中断							√	√
SPI中断		√		√	√	√	√	√
外部中断2 ($\overline{\text{INT2}}$)	√	√	√	√	√	√	√	√
外部中断3 ($\overline{\text{INT3}}$)	√	√	√	√	√	√	√	√
定时器2中断	√	√	√	√	√	√	√	√
外部中断4 ($\overline{\text{INT4}}$)	√	√	√	√	√	√	√	√
串口3中断								√
串口4中断								√
定时器3中断								√
定时器4中断								√
比较器中断			√	√	√	√		√

上表中 √ 表示对应的系列有相应的中断源。

6.1.1 STC15F101W系列单片机的中断请求源

STC15F101W系列单片机提供了8个中断请求源，它们分别是：外部中断0(INT0)、定时器0中断、外部中断1(INT1)、低压检测(LVD)中断、外部中断2($\overline{\text{INT2}}$)、外部中断3($\overline{\text{INT3}}$)、定时器2中断以及外部中断4($\overline{\text{INT4}}$)。除外部中断2($\overline{\text{INT2}}$)、外部中断3($\overline{\text{INT3}}$)、定时器2中断及外部中断4($\overline{\text{INT4}}$)固定是最低优先级中断外，其它的中断都具有两个中断优先级。

6.1.2 STC15F408AD系列单片机的中断请求源

STC15F408AD系列单片机提供了12个中断请求源，它们分别是：外部中断0(INT0)、定时器0中断、外部中断1(INT1)、串口中断、A/D转换中断、低压检测(LVD)中断、CCP/PWM/PCA中断、SPI中断、外部中断2($\overline{\text{INT2}}$)、外部中断3($\overline{\text{INT3}}$)，定时器2中断以及外部中断4($\overline{\text{INT4}}$)。除外部中断2($\overline{\text{INT2}}$)、外部中断3($\overline{\text{INT3}}$)、定时器T2中断及外部中断4($\overline{\text{INT4}}$)固定是最低优先级中断外，其它的中断都具有两个中断优先级。

6.1.3 STC15W201S系列单片机的中断请求源

STC15W201S系列单片机提供了10个中断请求源，它们分别是：外部中断0(INT0)、定时器0中断、外部中断1(INT1)、串口中断、低压检测(LVD)中断、外部中断2($\overline{\text{INT2}}$)、外部中断3($\overline{\text{INT3}}$)、定时器2中断、外部中断4($\overline{\text{INT4}}$)以及比较器中断。除外部中断2($\overline{\text{INT2}}$)、外部中断3($\overline{\text{INT3}}$)、定时器2中断、外部中断4($\overline{\text{INT4}}$)及比较器中断固定是最低优先级中断外，其它的中断都具有两个中断优先级。

STC MCU Limited

6.1.4 STC15W408AS系列单片机的中断请求源

STC15W408AS系列单片机提供了13个中断请求源，它们分别是：外部中断0(INT0)、定时器0中断、外部中断1(INT1)、串口中断、A/D转换中断、低压检测(LVD)中断、CCP/PWM/PCA中断、SPI中断、外部中断2(INT2)、外部中断3(INT3)、定时器2中断、外部中断4(INT4)及比较器中断。除外部中断2(INT2)、外部中断3(INT3)、定时器2中断、外部中断4(INT4)及比较器中断固定是最低优先级中断外，其它的中断都具有2个中断优先级。

6.1.5 STC15W408S系列单片机的中断请求源

STC15W408S系列单片机提供了12个中断请求源，它们分别是：外部中断0(INT0)、定时器0中断、外部中断1(INT1)、定时器1中断、串口中断、低压检测(LVD)中断、SPI中断、外部中断2(INT2)、外部中断3(INT3)、定时器2中断、外部中断4(INT4)及比较器中断。除外部中断2(INT2)、外部中断3(INT3)、定时器2中断、外部中断4(INT4)及比较器中断固定是最低优先级中断外，其它的中断都具有两个中断优先级。

6.1.6 STC15W1K16S系列单片机的中断请求源

STC15W1K16S系列单片机提供了12个中断请求源，它们分别是：外部中断0(INT0)、定时器0中断、外部中断1(INT1)、定时器1中断、串口中断、低压检测(LVD)中断、SPI中断、外部中断2(INT2)、外部中断3(INT3)，定时器2中断、外部中断4(INT4)及比较器中断。除外部中断2(INT2)、外部中断3(INT3)、定时器2中断、外部中断4(INT4)及比较器中断固定是最低优先级中断外，其它的中断都具有2个中断优先级，可实现2级中断服务程序嵌套。

6.1.7 STC15F2K60S2系列单片机的中断请求源

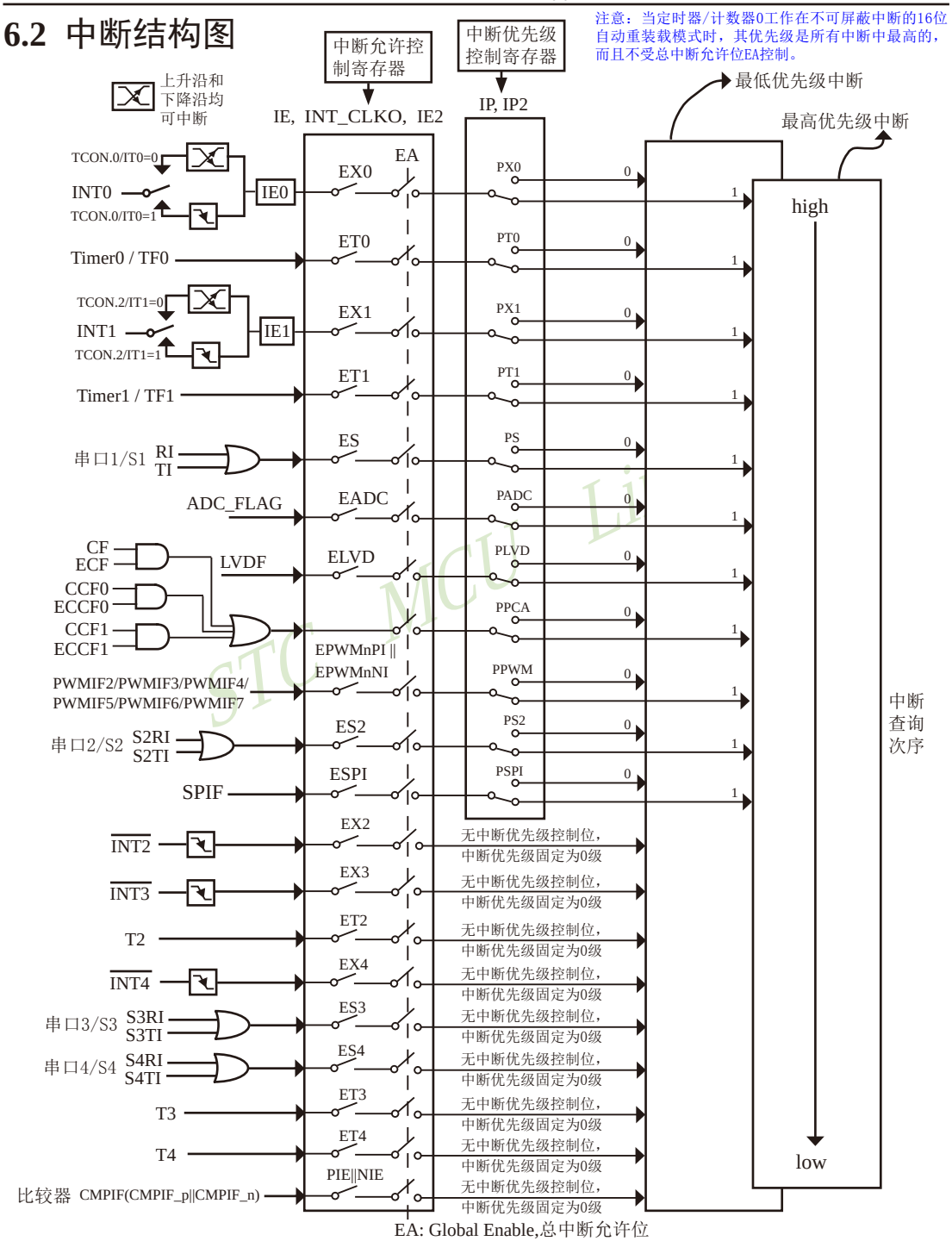
STC15F2K60S2系列单片机提供了14个中断请求源，它们分别是：外部中断0(INT0)、定时器0中断、外部中断1(INT1)、定时器1中断、串口1中断、A/D转换中断、低压检测(LVD)中断、CCP/PWM/PCA中断、串口2中断、SPI中断、外部中断2(INT2)、外部中断3(INT3)，定时器2中断以及外部中断4(INT4)。除外部中断2(INT2)、外部中断3(INT3)、定时器2中断及外部中断4(INT4)固定是最低优先级中断外，其它的中断都具有2个中断优先级，可实现2级中断服务程序嵌套。

6.1.8 STC15W4K60S4系列单片机的中断请求源

STC15W4K60S4系列单片机提供了20个中断请求源，它们分别是：外部中断0(INT0)、定时器0中断、外部中断1(INT1)、定时器1中断、串口1中断、A/D转换中断、低压检测(LVD)中断、CCP/PWM/PCA中断、串口2中断、SPI中断、外部中断2(INT2)、外部中断3(INT3)、定时器2中断、PWM中断、外部中断4(INT4)、串口3中断、串口4中断、定时器3中断、定时器4中断及比较器中断。除外部中断2(INT2)、外部中断3(INT3)、定时器2中断、外部中断4(INT4)、串口3中断、串口4中断、定时器3中断、定时器4中断及比较器中断固定是最低优先级中断外，其它的中断都具有2个中断优先级。

STC MCU Limited

6.2 中断结构图


注意：当定时器/计数器0工作在不可屏蔽中断的16位自动重装载模式时，其优先级是所有中断中最高的，而且不受总中断允许位EA控制。
中断允许控制寄存器
中断优先级控制寄存器
IE, INT_CLKO, IE2
IP, IP2
上升沿和下降沿均可中断
最低优先级中断
最高优先级中断
high
low
中断查询次序
TCON.0/IT0=0
INT0
TCON.0/IT0=1
IE0
EX0
EA
PX0
Timer0 / TF0
ET0
PT0
TCON.2/IT1=0
INT1
TCON.2/IT1=1
IE1
EX1
PX1
Timer1 / TF1
ET1
PT1
串口1/S1
RI
TI
ES
PS
ADC_FLAG
EADC
PADC
CF
ECF
LVDF
ELVD
PLVD
CCF0
ECCF0
CCF1
ECCF1
PPCA
EPWMnPI ||
EPWMnNI
PWMIF2/PWMIF3/PWMIF4/
PWMIF5/PWMIF6/PWMIF7
PPWM
串口2/S2
S2RI
S2TI
ES2
PS2
SPIF
ESPI
PSPI
INT2
EX2
INT3
EX3
T2
ET2
INT4
EX4
串口3/S3
S3RI
S3TI
ES3
串口4/S4
S4RI
S4TI
ES4
T3
ET3
T4
ET4
比较器 CMPIF(CMPIF_p||CMPIF_n)
PIE||NIE
无中断优先级控制位，
中断优先级固定为0级
0
1
EA: Global Enable,总中断允许位

外部中断0(INT0)和外部中断1(INT1)既可上升沿触发，又可下降沿触发。请求两个外部中断的标志位是位于寄存器TCON中的IE0/TCON.1和IE1/TCON.3。当外部中断服务程序被响应后，中断标志位IE0和IE1会自动被清0。TCON寄存器中的IT0/TCON.0和IT1/TCON.2决定了外部中断0和1是上升沿触发还是下降沿触发。如果ITx = 0(x = 0,1)，那么系统在INTx(x = 0,1)脚探测到上升沿或下降沿后均可产生外部中断。如果ITx = 1(x = 0,1)，那么系统在INTx(x= 0,1)脚探测下降沿后才可产生外部中断。外部中断0(INT0)和外部中断1(INT1)还可以用于将单片机从掉电模式唤醒。

定时器0和1的中断请求标志位是TF0和TF1。当定时器寄存器THx/TLx(x = 0,1)溢出时，溢出标志位TFx(x = 0,1)会被置位，如果定时器0/1的中断被打开，则定时器中断发生。当单片机转去执行该定时器中断时，定时器的溢出标志位TFx(x = 0,1)会被硬件清除。

外部中断2(INT2)、外部中断3(INT3)及外部中断4(INT4)都只能下降沿触发。外部中断2~4的中断请求标志位被隐藏起来了，对用户不可见。当相应的中断服务程序被响应后或EXn=0(n=2,3,4)，这些中断请求标志位会立即自动地被清0。外部中断2(INT2)、外部中断3(INT3)及外部中断4(INT4)也可以用于将单片机从掉电模式唤醒。

定时器2的中断请求标志位被隐藏起来了，对用户不可见。当相应的中断服务程序被响应后或ET2=0，该中断请求标志位会立即自动地被清0。

定时器3和定时器4的中断请求标志位同样被隐藏起来了，对用户不可见。当相应的中断服务程序被响应后或ET3=0 / ET4=0，该中断请求标志位会立即自动地被清0。

当串行口1发送或接收完成时，相应的中断请求标志位TI或RI就会被置位，如果串口1中断被打开，向CPU请求中断，单片机转去执行该串口1中断。中断响应后，TI或RI需由软件清零。

当串行口2发送或接收完成时，相应的中断请求标志位S2TI或S2RI就会被置位，如果串口2中断被打开，向CPU请求中断，则单片机转去执行该串口2中断。中断响应后，S2TI或S2RI需由软件清零。

当串行口3发送或接收完成时，相应的中断请求标志位S3TI或S3RI就会被置位，如果串口3中断被打开，向CPU请求中断，则单片机转去执行该串口3中断。中断响应后，S3TI或S3RI需由软件清零。

当串行口4发送或接收完成时，相应的中断请求标志位S4TI或S4RI就会被置位，如果串口4中断被打开，向CPU请求中断，则单片机转去执行该串口4中断。中断响应后，S4TI或S4RI需由软件清零。

A/D转换的中断是由ADC_FLAG/ADC_CONTR.4请求产生的。该位需用软件清除。

低压检测(LVD)中断是由LVDF/PCON.5请求产生的。该位也需用软件清除。

当同步串行口SPI传输完成时，SPIF/SPCTL.7被置位，如果SPI中断被打开，则向CPU请求中断，单片机转去执行该SPI中断。中断响应完成后，SPIF需通过软件向其写入“1”清零。

比较器中断标志位CMPIF=(CMPIF_p || CMPIF_n)，其中CMPIF_p是内建的标志比较器上升沿中断的寄存器，CMPIF_n是内建的标志比较器下降沿中断的寄存器；当CPU 去读取 CMPIF

的数值时会读到 (CMPIF_p || CMPIF_n)；当CPU 对CMPIF写“0”后CMPIF_p及CMPIF_n会被自动设置为“0”。因此，当比较器的比较结果由LOW变成HIGH时，那么内建的标志比较器上升沿中断的寄存器CMPIF_p会被设置成1，即比较器中断标志位CMPIF也会被设置成1，如果比较器上升沿中断已被允许，即PIE(CMPCR1.5)已被设置成1，则向CPU请求中断，单片机转去执行该比较器上升中断；同理，当比较器的比较结果由HIGH变成LOW时，那么内建的标志比较器下降沿中断的寄存器CMPIF_n会被设置成1，即比较器中断标志位CMPIF也会被设置成1，如果比较器下降沿中断已被允许，即NIE(CMPCR1.4)已被设置成1，则向CPU请求中断，单片机转去执行该比较器下降中断。中断响应完成后，比较器中断标志位CMPIF不会自动被清零，用户需通过软件向其写入“0”清零它。

各个中断触发行为总结如下表所示：

中断触发表

中断源	触发行为
INT0 (外部中断0)	(IT0 = 1): 下降沿；　(IT0 = 0): 上升沿和下降沿均可
Timer 0	定时器0溢出
INT1 (外部中断1)	(IT1 = 1): 下降沿；　(IT1 = 0): 上升沿和下降沿均可
Timer1	定时器1溢出
UART1	串口1发送或接收完成
ADC	A/D转换完成
LVD	电源电压下降到低于LVD检测电压
UART2	串口2发送或接收完成
SPI	SPI数据传输完成
$\overline{\text{INT2}}$ (外部中断2)	下降沿
$\overline{\text{INT3}}$ (外部中断3)	下降沿
Timer2	定时器2溢出
$\overline{\text{INT4}}$ (外部中断4)	下降沿
UART3	串口3发送或接收完成
UART4	串口4发送或接收完成
Timer3	定时器3溢出
Timer4	定时器4溢出
Comparator (比较器)	比较器比较结果由LOW变成HIHG或由HIGH变成LOW

STC MCU Limited

6.3 中断向量入口地址/查询次序/优先级/请求标志/允许位表

中断向量入口地址/查询次序/优先级/请求标志位/允许位

中断源	中断向量地址	相同优先级内的查询次序	中断优先级设置	优先级0（最低）	优先级1（最高）	中断请求标志位		中断允许控制位
INT0 (外部中断 0)	0003H	0 (highest)	PX0	0	1	IE0		EX0/EA
Timer 0	000BH	1	PT0	0	1	TF0		ET0/EA
INT1 (外部中断1)	0013H	2	PX1	0	1	IE1		EX1/EA
Timer1	001BH	3	PT1	0	1	TF1		ET1/EA
S1(UART1)	0023B	4	PS	0	1	RI+TI		ES/EA
ADC	002BH	5	PADC	0	1	ADC_FLAG		EADC/EA
LVD	0033H	6	PLVD	0	1	LVDF		ELVD/EA
CCP/PCA/PWM	003BH	7	PPCA	0	1	CF+CCF0+CCF1 +CCF2		(ECF+ECCF0+ECCF1 +ECCF2)/EA
S2(UART2)	0043H	8	PS2	0	1	S2RI+S2TI		ES2/EA
SPI	004BH	9	PSPI	0	1	SPIF		ESPI/EA
$\overline{\text{INT2}}$ (外部中断2)	0053H	10	0	0				EX2/EA
$\overline{\text{INT3}}$ (外部中断3)	005BH	11	0	0				EX3/EA
Timer 2	0063H	12	0	0				ET2/EA
PWM	006BH	13	PPWM	0	1	PWMIF2/3/4/5/6/7		EPWMnPI||EPWMnNI/ EA
System Reserved	0073H	14						
System Reserved	007BH	15						
$\overline{\text{INT4}}$ (外部中断4)	0083H	16	0	0				EX4/EA
S3(UART3)	008BH	17	0	0		S3RI+S3TI		ES3/EA
S4(UART4)	0093H	18	0	0		S4RI+S4TI		ES4/EA
Timer 3	009BH	19	0	0				ET3/EA
Timer 4	00A3H	20	0	0				ET4/EA
Comparator (比较器)	00ABH	21(lowest)	0	0		CMPIF	CMPIF_p	PIE/EA （比较器上升沿中断允许位）
							CMPIF_n	NIE/EA （比较器下降沿中断允许位）

STC MCU Limited

6.4 在Keil C中如何声明中断函数

如果使用C语言编程，中断查询次序号就是中断号，例如：

```
void    Int0_Routine(void)          interrupt  0;
void    Timer0_Rountine(void)       interrupt  1;
void    Int1_Routine(void)          interrupt  2;
void    Timer1_Rountine(void)       interrupt  3;
void    UART1_Routine(void)         interrupt 4;
void    ADC_Routine(void)           interrupt  5;
void    LVD_Routine(void)           interrupt  6;
void    PCA_Routine(void)           interrupt 7;
void    UART2_Routine(void)         interrupt 8;
void    SPI_Routine(void)           interrupt 9;
void    Int2_Routine(void)          interrupt  10;
void    Int3_Routine(void)          interrupt  11;
void    Timer2_Routine(void)        interrupt  12;
void    PWM_Routine(void)           interrupt  13;
void    Int4_Routine(void)          interrupt  16;
void    S3_Routine(void)            interrupt  17;
void    S4_Routine(void)            interrupt  18;
void    Timer3_Routine(void)        interrupt  19;
void    Timer4_Routine(void)        interrupt 20;
void    Comparator_Routine(void)    interrupt 21;
```

STC MCU Limited

6.5 中断寄存器

符号	描述	地址	位地址及符号 MSB LSB								复位值
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IE2	Interrupt Enable 2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
INT_CLKO AUXR2	外部中断允许和时钟输出寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000 x000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
IP2	2rd Interrupt Priority Low register	B5H	-	-	-	-	-	PPWM	PSPI	PS2	xxxx xx00B
TCON	Timer Control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
SCON	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
S2CON	Serial 2/ UART2 Control	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000 0000B
S3CON	串口3控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S4CON	串口4控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
T4T3M	T4和T3的控制寄存器	D1H	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO	0000 0000B
PCON	Power Control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B
ADC_CONTR	ADC control register	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHIS0	0000 0000B
SPSTAT	SPI Status register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx xxxxB
CCON	PCA Control Register	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx x000B
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF	0xxx 0000B
CCAPM0	PCA Module 0 Mode Register	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x000 0000B
CCAPM1	PCA Module 1 Mode Register	DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x000 0000B
CCAPM2	PCA Module 2 Mode Register	DCH	-	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	x000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
CMPCR1	比较器控制寄存器1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000 0000B
PWMIF	PWM中断标志寄存器	DFH	PWMIF7	PWMIF6	PWMIF5	PWMIF4	PWMIF3	PWMIF2			0000 0000B

STC 国芯 Limited

上表中列出了与STC15F2K60S2系列单片机中断相关的所有寄存器，下面逐一地对这些寄存器进行介绍。

1. 中断允许寄存器IE、IE2和INT_CLKO

STC15F2K60S2系列单片机CPU对中断源的开放或屏蔽，每一个中断源是否被允许中断，是由内部的中断允许寄存器IE（IE为特殊功能寄存器，它的字节地址为A8H）控制的，其格式如下：

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的总中断允许控制位，EA=1，CPU开放中断，EA=0，CPU屏蔽所有的中断申请。
EA的作用是使中断允许形成多级控制。即各中断源首先受EA控制;其次还受各中断源自己的中断允许控制位控制。

ELVD：低压检测中断允许位，ELVD=1，允许低压检测中断，ELVD=0，禁止低压检测中断。

EADC：A/D转换中断允许位，EADC=1，允许A/D转换中断，EADC=0，禁止A/D转换中断。

ES：串行口1中断允许位，ES=1，允许串行口1中断，ES=0，禁止串行口1中断。

ET1：定时/计数器T1的溢出中断允许位，ET1=1，允许T1中断，ET1=0，禁止T1中断。

EX1：外部中断1中断允许位，EX1=1，允许外部中断1中断，EX1=0，禁止外部中断1中断。

ET0：T0的溢出中断允许位，ET0=1允许T0中断，ET0=0禁止T0中断。

EX0：外部中断0中断允许位，EX0=1允许中断，EX0=0禁止中断。

IE2：中断允许寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4：定时器4的中断允许位。
ET4=1,允许定时器4产生中断;
ET4=0,禁止定时器4产生中断

ET3：定时器3的中断允许位。
ET3=1,允许定时器3产生中断;
ET3=0,禁止定时器3产生中断

ES4：串行口4中断允许位。
ES4=1，允许串行口4中断;
ES4=0，禁止串行口4中断

ES3：串行口3中断允许位。
ES3=1，允许串行口3中断;
ES3=0，禁止串行口3中断。。

STC MCU Limited

ET2：定时器2的中断允许位。
ET2=1,允许定时器2产生中断；
ET2=0,禁止定时器2产生中断

ESPI：SPI中断允许位。
ESPI=1，允许SPI中断；
ESPI=0，禁止SPI中断。

ES2：串行口2中断允许位。
ES2=1，允许串行口2中断；
ES2=0，禁止串行口2中断。

INT_CLKO (AUXR2)是STC15系列单片机新增寄存器，地址是8FH，INT_CLKO (AUXR2)格式如下：

INT_CLKO (AUXR2)：外部中断允许和时钟输出寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO (AUXR2)	8FH	name	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

EX4：外部中断4($\overline{INT4}$)中断允许位，EX4=1允许中断，EX4=0禁止中断。外部中断4($\overline{INT4}$)只能下降沿触发。

EX3：外部中断3($\overline{INT3}$)中断允许位，EX3=1允许中断，EX3=0禁止中断。外部中断3($\overline{INT3}$)也只能下降沿触发。

EX2：外部中断2($\overline{INT2}$)中断允许位，EX2=1允许中断，EX2=0禁止中断。外部中断2($\overline{INT2}$)同样只能下降沿触发。

T2CLKO, T1CLKO,T0CLKO与中断无关，在此不作介绍。

STC15系列单片机复位以后，IE、IE2和INT_CLKO(AUXR2)被清0，由用户程序置“1”或清“0”IE、IE2和INT_CLKO (AUXR2)的相应位，实现允许或禁止各中断源的中断申请，若使某一个中断源允许中断必须同时使CPU开放中断。更新IE的内容可由位操作指令来实现（SETB BIT；CLR BIT），也可用字节操作指令实现（即MOV IE，#DATA，ANL IE，#DATA；ORL IE，#DATA；MOV IE，A等）。更新IE2和INT_CLKO(不可位寻址)的内容只可用字节操作指令(即MOV IE2, #DATA或MOV INT_CLKO, #DATA)来解决。

STC MCU Limited

2. 中断优先级控制寄存器IP、IP2

传统8051单片机具有两个中断优先级，即高优先级和低优先级，可以实现两级中断嵌套。STC15系列单片机通过设置特殊功能寄存器(IP和IP2)中的相应位，可将部分中断设有2个中断优先级，除外部中断2($\overline{INT2}$)、外部中断3($\overline{INT3}$)及外部中断4($\overline{INT4}$)外，所有中断请求源可编程为2个优先级中断。一个正在执行的低优先级中断能被高优先级中断所中断，但不能被另一个低优先级中断所中断，一直执行到结束，遇到返回指令RETI，返回主程序后再执行一条指令才能响应新的中断申请。以上所述可归纳为下面两条基本规则：

1. 低优先级中断可被高优先级中断所中断，反之不能。
2. 任何一种中断(不管是高级还是低级)，一旦得到响应，不会再被它的同级中断所中断

STC15系列单片机的片内各优先级控制寄存器的格式如下:

IP：中断优先级控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PPCA:　PCA中断优先级控制位。
当PPCA=0时，PCA中断为最低优先级中断(优先级0)
当PPCA=1时，PCA中断为最高优先级中断(优先级1)

PLVD:　低压检测中断优先级控制位。
当PLVD=0时，低压检测中断为最低优先级中断(优先级0)
当PLVD=1时，低压检测中断为最高优先级中断(优先级1)

PADC:　A/D转换中断优先级控制位。
当PADC=0时，A/D转换中断为最低优先级中断(优先级0)
当PADC=1时，A/D转换中断为最高优先级中断(优先级1)

PS:　串口1中断优先级控制位。
当PS=0时，串口1中断为最低优先级中断(优先级0)
当PS=1时，串口1中断为最高优先级中断(优先级1)

PT1:　定时器1中断优先级控制位。
当PT1=0时，定时器1中断为最低优先级中断(优先级0)
当PT1=1时，定时器1中断为最高优先级中断(优先级1)

PX1:　外部中断1优先级控制位。
当PX1=0时，外部中断1为最低优先级中断(优先级0)
当PX1=1时，外部中断1为最高优先级中断(优先级1)

PT0:　定时器0中断优先级控制位。
当PT0=0时，定时器0中断为最低优先级中断(优先级0)
当PT0=1时，定时器0中断为最高优先级中断(优先级1)

PX0:　外部中断0优先级控制位。
当PX0=0时，外部中断0为最低优先级中断(优先级0)
当PX0=1时，外部中断0为最高优先级中断(优先级1)

STC MCU Limited

IP2：中断优先级控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP2	B5H	name	-	-	-	-	-	-	PSPI	PS2

PSPI: SPI中断优先级控制位。

当PSPI=0时，SPI中断为最低优先级中断(优先级0)

当PSPI=1时，SPI中断为最高优先级中断(优先级1)

PS2: 串口2中断优先级控制位。

当PS2=0时，串口2中断为最低优先级中断(优先级0)

当PS2=1时，串口2中断为最高优先级中断(优先级1)

中断优先级控制寄存器IP和IP2的各位都由可用户程序置“1”和清“0”。但IP寄存器可位操作，所以可用位操作指令或字节操作指令更新IP的内容。而IP2寄存器的内容只能用字节操作指令来更新。STC15系列单片机复位后IP和IP2均为00H，各个中断源均为低优先级中断。

3. 定时器/计数器控制寄存器TCON

TCON为定时器/计数器T0、T1的控制寄存器，同时也锁存T0、T1溢出中断源和外部请求中断源等，TCON格式如下：

TCON：定时器/计数器中断控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	name	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：T1溢出中断标志。T1被允许计数以后，从初值开始加1计数。当产生溢出时由硬件置“1”TF1，向CPU请求中断，一直保持到CPU响应中断时，才由硬件清“0”（也可由查询软件清“0”）。

TR1：定时器1的运行控制位。

TF0：T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

TR0：定时器0的运行控制位。

IE1：外部中断1（INT1/P3.3）中断请求标志。IE1=1，外部中断向CPU请求中断，当CPU响应该中断时由硬件清“0”IE1。

IT1：外部中断1中断源类型选择位。IT1=0，INT1/P3.3引脚上的上升沿或下降沿信号均可触发外部中断1。IT1=1，外部中断1为下降沿触发方式。

IE0：外部中断0（INT0/P3.2）中断请求标志。IE0=1，外部中断0向CPU请求中断，当CPU响应外部中断时，由硬件清“0”IE0。

IT0：外部中断0中断源类型选择位。IT0=0，INT0/P3.2引脚上的上升沿或下降沿均可触发外部中断0。IT0=1，外部中断0为下降沿触发方式。

STC MCU Limited

4. 串行口1控制寄存器SCON

SCON为串行口控制寄存器，SCON格式如下：

SCON：串行口控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

RI：　串行口1接收中断标志。若串行口1允许接收且以方式0工作，则每当接收到第8位数据时置1；若以方式1、2、3工作且SM2=0时，则每当接收到停止位的中间时置1；当串行口以方式2或方式3工作且SM2=1时，则仅当接收到的第9位数据RB8为1后，同时还要接收到停止位的中间时置1。RI为1表示串行口1正向CPU申请中断(接收中断)，RI必须由用户的中断服务程序清零。

TI：　串行口1发送中断标志。串行口1以方式0发送时，每当发送完8位数据，由硬件置1；若以方式1、方式2或方式3发送时，在发送停止位的开始时置1。TI=1表示串行口1正在向CPU申请中断(发送中断)。值得注意的是，CPU响应发送中断请求，转向执行中断服务程序时并不将TI清零，TI必须由用户在中断服务程序中清零。

SCON寄存器的其他位与中断无关，在此不作介绍。

5. 串行口2控制寄存器S2CON

S2CON为串行口2控制寄存器，S2CON格式如下：

S2CON：串行口2控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	name	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2RI：串行口2接收中断标志。若串行口2允许接收且以方式0工作，则每当接收到第8位数据时置1；若以方式1、2、3工作且S2SM2=0时，则每当接收到停止位的中间时置1；当串行口2以方式2或方式3工作且S2SM2=1时，则仅当接收到的第9位数据S2RB8为1后，同时还要接收到停止位的中间时置1。S2RI为1表示串行口2正向CPU申请中断(接收中断)，S2RI必须由用户的中断服务程序清零。

S2TI：串行口2发送中断标志。串行口2以方式0发送时，每当发送完8位数据，由硬件置1；若以方式1、方式2或方式3发送时，在发送停止位的开始时置1。S2TI=1表示串行口2正在向CPU申请中断(发送中断)。值得注意的是，CPU响应发送中断请求，转向执行中断服务程序时并不将S2TI清零，S2TI必须由用户在中断服务程序中清零。

S2CON寄存器的其他位与中断无关，在此不作介绍。

STC MCU Limited

6. 串行口3控制寄存器S3CON

S3CON为串行口3控制寄存器，S3CON格式如下：

S3CON：串行口3控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	name	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3RI：串行口3接收中断标志。若串行口3允许接收且以方式0工作，则每当接收到第8位数据时置1；若以方式1、2、3工作且S3SM2=0时，则每当接收到停止位的中间时置1；当串行口3以方式2或方式3工作且S3SM2=1时，则仅当接收到的第9位数据S3RB8为1后，同时还要接收到停止位的中间时置1。S3RI为1表示串行口3正向CPU申请中断(接收中断)，S3RI必须由用户的中断服务程序清零。

S3TI：串行口3发送中断标志。串行口3以方式0发送时，每当发送完8位数据，由硬件置1；若以方式1、方式2或方式3发送时，在发送停止位的开始时置1。S3TI=1表示串行口3正在向CPU申请中断(发送中断)。值得注意的是，CPU响应发送中断请求，转向执行中断服务程序时并不将S3TI清零，S3TI必须由用户在中断服务程序中清零。

S3CON寄存器的其他位与中断无关，在此不作介绍。

7. 串行口4控制寄存器S4CON

S4CON为串行口4控制寄存器，S4CON格式如下：

S4CON：串行口4控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	name	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4RI：串行口4接收中断标志。若串行口4允许接收且以方式0工作，则每当接收到第8位数据时置1；若以方式1、2、3工作且S4SM2=0时，则每当接收到停止位的中间时置1；当串行口4以方式2或方式3工作且S4SM2=1时，则仅当接收到的第9位数据S4RB8为1后，同时还要接收到停止位的中间时置1。S4RI为1表示串行口4正向CPU申请中断(接收中断)，S4RI必须由用户的中断服务程序清零。

S4TI：串行口4发送中断标志。串行口4以方式0发送时，每当发送完8位数据，由硬件置1；若以方式1、方式2或方式3发送时，在发送停止位的开始时置1。S4TI=1表示串行口4正在向CPU申请中断(发送中断)。值得注意的是，CPU响应发送中断请求，转向执行中断服务程序时并不将S4TI清零，S4TI必须由用户在中断服务程序中清零。

S4CON寄存器的其他位与中断无关，在此不作介绍。

STC MCU Limited

8. 低压检测中断相关寄存器：电源控制寄存器PCON

PCON为电源控制寄存器，PCON格式如下：

PCON：电源控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF：低压检测标志位,同时也是低压检测中断请求标志位。

在正常工作和空闲工作状态时，如果内部工作电压Vcc低于低压检测门槛电压，该位自动置1，与低压检测中断是否被允许无关。即在内部工作电压Vcc低于低压检测门槛电压时，不管有没有允许低压检测中断，该位都自动为1。该位要用软件清0，清0后，如内部工作电压Vcc继续低于低压检测门槛电压，该位又被自动设置为1。

在进入掉电工作状态前，如果低压检测电路未被允许可产生中断，则在进入掉电模式后，该低压检测电路不工作以降低功耗。如果被允许可产生低压检测中断，则在进入掉电模式后，该低压检测电路继续工作，在内部工作电压Vcc低于低压检测门槛电压后，产生低压检测中断，可将MCU从掉电状态唤醒。

电源控制寄存器PCON中的其他位与低压检测中断无关，在此不作介绍。

在中断允许寄存器IE中，低压检测中断相应的允许位是ELVD/IE.6

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的总中断允许控制位，EA=1，CPU开放中断，EA=0，CPU屏蔽所有的中断申请。EA的作用是使中断允许形成两级控制。即各中断源首先受EA控制;其次还受各中断源自己的中断允许控制位控制。

ELVD：低压检测中断允许位，ELVD=1，允许低压检测中断，ELVD=0，禁止低压检测中断。

9. A/D转换控制寄存器ADC_CONTR

ADC_CONTR为A/D转换控制寄存器，ADC_CONTR格式如下：

ADC_CONTR：A/D转换控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	name	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

ADC_POWER：ADC电源控制位。当ADC_POWER=0时，关闭ADC电源;
当ADC_PWOER=1时，打开ADC电源。

ADC_FLAG：ADC转换结束标志位，可用于请求A/D转换的中断。当A/D转换完成后，ADC_FLAG=1,要用软件清0。不管是A/D转换完成后由该位申请产生中断，还是由软件查询该标志位A/D转换是否结束，当A/D转换完成后，ADC_FLAG=1，一定要软件清0。

ADC_START：ADC转换启动控制位，设置为“1”时,开始转换，转换结束后为0。

A/D转换控制寄存器ADC_CONTR中的其他位与中断无关，在此不作介绍。

在中断允许寄存器IE中，A/D转换器的中断允许位是EADC/IE.5

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的总中断允许控制位，EA=1，CPU开放中断，EA=0，CPU屏蔽所有的中断申请。EA的作用是使中断允许形成两级控制。即各中断源首先受EA控制;其次还受各中断源自己的中断允许控制位控制。

EADC：A/D转换中断允许位，EADC=1，允许A/D转换中断，EADC=0，禁止A/D转换中断。

10. 比较器控制寄存器1：CMPCR1

比较器控制寄存器1的格式如下：

CMPCR1：比较器控制寄存器1

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	name	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPEN：比较器模块使能位

CMPEN=1，使能比较器模块;

CMPEN=0，禁用比较器模块,比较器的电源关闭。

CMPIF：比较器中断标志位(Interrupt Flag)

在 CMPEN为1的情况下：

当比较器的比较结果由LOW变成HIGH时, 若是PIE被设置成1, 那么内建的某一个叫做CMPIF_p的寄存器会被设置成1;

当比较器的比较结果由HIGH变成LOW时, 若是NIE被设置成1, 那么内建的某一个叫做CMPIF_n的寄存器会被设置成1;

当CPU 去读取 CMPIF 的数值时, 会读到 (CMPIF_p || CMPIF_n);

当CPU对CMPIF写0后, CMPIF_p 以及 CMPIF_n 都会被清除为0 .

而中断产生的条件是 [(EA==1) && (((PIE==1)&&(CMPIF_p==1)) || ((NIE==1)&&(CMPIF_n==1)))]

CPU接受中断后, 并不会自动清除此CMPIF标志, 用户必须用软件写”0”去清除它。

PIE：比较器上升沿中断使能位(Pos-edge Interrupt Enabling)

PIE = 1，使能比较器由LOW变HIGH的事件 设定CMPIF_p产生中断;

PIE = 0，禁用比较器由LOW变HIGH的事件 设定CMPIF_p产生中断。

NIE：比较器下降沿中断使能位 (Neg-edge Interrupt Enabling)

NIE = 1，使能比较器由HIGH变LOW的事件 设定CMPIF_n产生中断;

NIE = 0，禁用比较器由HIGH变LOW的事件 设定CMPIF_n产生中断。

比较器控制寄存器1—CMPCR1 的其他位与中断无关，在此不作介绍。

STC MCU Limited

6.6 中断优先级

除外部中断2(INT2)、外部中断3(INT3)、定时器T2中断、外部中断4(INT4)、串口3中断、串口4中断、定时器T3中断、定时器T4中断及比较器中断外，STC15W4K60S4系列单片机的所有的中断都具有2个中断优先级。一个正在执行的低优先级中断能被高优先级中断所中断，但不能被另一个低优先级中断所中断，一直执行到结束，遇到返回指令RETI，返回主程序后再执行一条指令才能响应新的中断申请。以上所述可归纳为下面两条基本规则：

1. 低优先级中断可被高优先级中断所中断，反之不能。

2. 任何一种中断（不管是高级还是低级），一旦得到响应，不能被它的同级中断所中断。

当同时收到几个同一优先级的中断要求时，哪一个要求得到服务，取决于内部的查询次序。这相当于在每个优先级内，还同时存在另一个辅助优先级结构，STC15W4K60S4系列单片机各中断优先查询次序如下：

	中断源	查询次序
0.	INT0	(highest)
1.	Timer 0	
2.	INT1	
3.	Timer 1	
4.	UART1	
5.	ADC interrupt	
6.	LVD	
7.	PCA	
8.	UART2	
9.	SPI	
10.	INT2	
11.	INT3	
12.	Timer 2	
13.		
14.		
15.		
16.	INT4	
17.	UART3	
18.	UART4	
19.	Timer 3	
20.	Timer 4	
21.	Comparator	(lowest)

STC MCU Limited

注意：当定时器/计数器0工作在不可屏蔽中断的16位自动重装载模式时，如果此时定时器/计数器0中断被允许了(只需置位ET0即可，不需置位EA,工作在不可屏蔽中断的16位自动重装载模式下的定时器/计数器0中断与EA无关)，则该中断优先级是所有中断中最高的，任何一个中断都不能打断它，而且该中断被打开后它不仅不受EA控制也不再受ET0控制了。

如果使用C 语言编程，中断查询次序号就是中断号，例如：

```
void    Int0_Routine(void)          interrupt  0;
void    Timer0_Rountine(void)       interrupt  1;
void    Int1_Routine(void)          interrupt  2;
void    Timer1_Rountine(void)       interrupt  3;
void    UART1_Routine(void)         interrupt 4;
void    ADC_Routine(void)           interrupt  5;
void    LVD_Routine(void)           interrupt  6;
void    PCA_Routine(void)           interrupt 7;
void    UART2_Routine(void)         interrupt 8;
void    SPI_Routine(void)           interrupt 9;
void    Int2_Routine(void)          interrupt  10;
void    Int3_Routine(void)          interrupt  11;
void    Timer2_Routine(void)        interrupt  12;
void    Int4_Routine(void)          interrupt  16;
void    S3_Routine(void)            interrupt  17;
void    S4_Routine(void)            interrupt  18;
void    Timer3_Routine(void)        interrupt  19;
void    Timer4_Routine(void)        interrupt 20;
void    Comparator_Routine(void)    interrupt 21;
```

STC MCU Limited

6.7 中断处理

当某中断产生而且被CPU响应，主程序被中断，接下来将执行如下操作：

1. 当前正被执行的指令全部执行完毕；
2. PC值被压入栈；
3. 现场保护；
4. 阻止同级别其他中断；
5. 将中断向量地址装载到程序计数器PC；
6. 执行相应的中断服务程序。

中断服务程序ISR完成和该中断相应的一些操作。中断服务程序ISR以RETI(中断返回)指令结束，将PC值从栈中取回，并恢复原来的中断设置，之后从主程序的断点处继续执行。

当某中断被响应时，被装载到程序计数器PC中的数值称为中断向量，是该中断源相对应的中断服务程序的起始地址。各中断源服务程序的入口地址（即中断向量）为：

中断源	中断向量
External Interrupt 0	0003H
Timer 0	000BH
External Interrupt 1	0013H
Timer 1	001BH
S1(UART1)	0023H
ADC interrupt	002BH
LVD	0033H
PCA	003BH
S2(UART2)	0043H
SPI	004BH
External Interrupt 2	0053H
External Interrupt 3	005BH
Timer 2	0063H
/	006BH
/	0073H
/	007BH
External Interrupt 4	0083H
S3(UART3)	008BH
S4(UART4)	0093H
Timer 3	009BH
Timer 4	00A3H
Comparator	00ABH

STC MCU Limited

当“转去执行中断”时，引起外部中断INT0/INT1/$\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$请求标志位和定时器/计数器0、定时器/计数器1的中断请求标志位将被硬件自动清零，其它中断的中断请求标志位需软件清″0″。由于中断向量入口地址位于程序存储器的开始部分，所以主程序的第1条指令通常为跳转指令，越过中断向量区(LJMP　MAIN)。

注意:不能用RET指令代替RETI指令

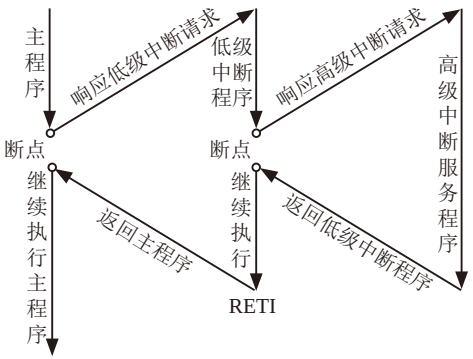
RET指令虽然也能控制PC返回到原来中断的地方，但RET指令没有清零中断优先级状态触发器的功能，中断控制系统会认为中断仍在进行，其后果是与此同级或低级的中断请求将不被响应。

若用户在中断服务程序中进行了入栈操作，则在RETI指令执行前应进行相应的出栈操作，即在中断服务程序中PUSH指令与POP指令必须成对使用，否则不能正确返回断点。

STC MCU Limited

6.8 中断嵌套

中断优先级高的中断请求可以中断CPU正在处理的优先级更低的中断服务程序，待完成了中断优先权高的中断服务程序后，再继续被打断的更低的中断服务程序。这就是中断嵌套。下图描述了主程序和中断服务程序运行示意图。



6.9 外部中断

外部中断0(INT0)和外部中断1(INT1)触发有两种触发方式，上升沿或下降沿均可触发方式和仅下降沿触发方式。

TCON寄存器中的IT0/TCON.0和IT1/TCON.2决定了外部中断0和1是上升沿和下降沿均可触发还是仅下降沿触发。如果ITx = 0(x = 0,1)，那么系统在INTx(x = 0,1)脚探测到上升沿或下降沿后均可产生外部中断。如果ITx = 1(x = 0,1)，那么系统在INTx(x= 0,1)脚探测下降沿后才可产生外部中断。外部中断0(INT0)和外部中断1(INT1)还可以用于将单片机从掉电模式唤醒。

外部中断2(INT2)、外部中断3(INT3)及外部中断4(INT4)都只能下降沿触发。外部中断2~4的中断请求标志位被隐藏起来了，对用户不可见，故也无需用户清"0"。当相应的中断服务程序被响应后或中断允许位EXn (n=2,3,4)被清零后，这些中断请求标志位会立即自动地被清0。这些中断请求标志位也可以通过软件禁止相应的中断允许控制位将其清“0”(特殊应用)。外部中断2(INT2)、外部中断3(INT3)及外部中断4(INT4)也可以用于将单片机从掉电模式唤醒。

由于系统每个时钟对外部中断引脚采样1次，所以为了确保被检测到，输入信号应该至少维持2个时钟。如果外部中断是仅下降沿触发，要求必须在相应的引脚维持高电平至少1个时钟，而且低电平也要持续至少一个时钟，才能确保该下降沿被CPU检测到。同样，如果外部中断是上升沿、下降沿均可触发，则要求必须在相应的引脚维持低电平或高电平至少1个时钟，而且高电平或低电平也要持续至少一个时钟，这样才能确保CPU能够检测到该上升沿或下降沿。

STC15系列单片机的3路CCP/PCA/PWM还再可实现3个外部中断(支持上升沿/下降沿中断)

STC MCU Limited

6.10 中断的测试程序(C和汇编)

6.10.1 外部中断0(INT0)的测试程序

6.10.1.1 外部中断INT0(上升沿+下降沿)的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT0中断举例--------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------

bit     FLAG;                   //1:上升沿中断 0:下降沿中断
sbit    P10     =       P1^0;

//-----------------------------------------
//外部中断服务程序
void exint0() interrupt 0               //INT0中断入口
{
        P10     =       !P10;           //将测试口取反
        FLAG    =       INT0;           //保存INT0口的状态, INT0=0(下降沿); INT0=1(上升沿)
}

//-----------------------------------------------
void main()
{
        INT0    =       1;
        IT0     =       0;              //设置INT0的中断类型 (1:仅下降沿 0:上升沿和下降沿)
        EX0     =       1;              //使能INT0中断
        EA      =       1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT0中断举例--------------------------------------------------------*/
/* --- 研发顾问QQ：  800003751---------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

FLAG    BIT     20H.0                   //1:上升沿中断 0:下降沿中断
//---------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     0003H                   //INT0中断入口
        LJMP    EXINT0
//---------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        CLR     IT0                     //设置INT0的中断类型 (1:仅下降沿 0:上升沿和下降沿)
        SETB    EX0                     //使能INT0中断
        SETB    EA
        SJMP    $

//---------------------------------------
//外部中断服务程序

EXINT0:
        CPL     P1.0                    //将测试口取反
        PUSH    PSW
        MOV     C,      INT0            //读取INT0口的状态
        MOV     FLAG,   C               //保存, INT0=0(下降沿); INT0=1(上升沿)
        POP     PSW
        RETI
;---------------------------------------

        END
```

STC MCU Limited

6.10.1.2 外部中断INT0(下降沿)的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT0中断举例--------------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------

sbit    P10     =       P1^0;

//-----------------------------------------
//外部中断服务程序
void exint0() interrupt 0                       //INT0中断入口
{
        P10     =       !P10;                   //将测试口取反
}

//-----------------------------------------

void main()
{
        INT0    =       1;
        IT0     =       1;                      //设置INT0的中断类型 (1:仅下降沿 0:上升沿和下降沿)
        EX0     =       1;                      //使能INT0中断
        EA      =       1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT0中断举例--------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     0003H                   //INT0中断入口
        LJMP    EXINT0

//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        SETB    IT0                     //设置INT0的中断类型 (1:仅下降沿 0:上升沿和下降沿)
        SETB    EX0                     //使能INT0中断
        SETB    EA
        SJMP    $

//-----------------------------------------
//外部中断服务程序

EXINT0:
        CPL     P1.0                    //将测试口取反
        RETI

;-----------------------------------------

        END
```

STC MCU Limited

6.10.2 外部中断1(INT1)的测试程序

6.10.2.1 外部中断INT1(上升沿+下降沿)的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT1中断举例--------------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------

bit     FLAG;                   //1:上升沿中断 0:下降沿中断
sbit    P10     =   P1^0;

//-----------------------------------------
//外部中断服务程序
void exint1() interrupt 2               //INT1中断入口
{
        P10     =       !P10;           //将测试口取反
        FLAG    =       INT1;           //保存INT1口的状态, INT1=0(下降沿); INT1=1(上升沿)
}

//-----------------------------------------------
void main()
{
        INT1    =       1;
        IT1     =       0;              //设置INT1的中断类型 (1:仅下降沿 0:上升沿和下降沿)
        EX1     =       1;              //使能INT1中断
        EA      =       1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT1中断举例--------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

FLAG    BIT     20H.0                   //1:上升沿中断 0:下降沿中断

//---------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     0013H                   //INT1中断入口
        LJMP    EXINT1

//---------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        CLR     IT1                     //设置INT1的中断类型 (1:仅下降沿 0:上升沿和下降沿)
        SETB    EX1                     //使能INT1中断
        SETB    EA
        SJMP    $

//---------------------------------------
//外部中断服务程序

EXINT1:
        CPL     P1.0                    //将测试口取反
        PUSH    PSW
        MOV     C,      INT1            //读取INT1口的状态
        MOV     FLAG,   C               //保存, INT1=0(下降沿); INT0=1(上升沿)
        POP     PSW
        RETI
;---------------------------------------

        END
```

STC MCU Limited

6.10.2.2 外部中断INT1(下降沿)的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT1中断举例--------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------

sbit    P10     =       P1^0;

//-----------------------------------------------
//外部中断服务程序
void exint1() interrupt 2                       //INT1中断入口
{
        P10     =       !P10;                   //将测试口取反
}

//-----------------------------------------------

void main()
{
        INT1    =       1;
        IT1     =       1;                      //设置INT1的中断类型 (1:仅下降沿 0:上升沿和下降沿)
        EX1     =       1;                      //使能INT1中断
        EA      =       1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 INT1中断举例--------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     0013H                   //INT1中断入口
        LJMP    EXINT1

//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        SETB    IT1                     //设置INT1的中断类型 (1:仅下降沿 0:上升沿和下降沿)
        SETB    EX1                     //使能INT1中断
        SETB    EA
        SJMP    $

//-----------------------------------------
//外部中断服务程序

EXINT1:
        CPL     P1.0                    //将测试口取反
        RETI

;-----------------------------------------

        END
```

STC MCU Limited

6.10.3 外部中断2(INT2)(下降沿中断)的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------*/
/* --- 演示STC 15 系列单片机外部中断2 (INT2) (下降沿) ----------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------

sfr     INT_CLKO    =   0x8f;       //外部中断与时钟输出控制寄存器
sbit    P10     =   P1^0;

//-----------------------------------------------
//外部中断服务程序
void exint2() interrupt 10                  //INT2中断入口
{
    P10     =   !P10;                       //将测试口取反

//  INT_CLKO    &=  0xEF;       //若需要手动清除中断标志,可先关闭中断,
                                //此时系统会自动清除内部的中断标志
//  INT_CLKO    |=  0x10;       //然后再开中断即可
}

void main()
{
    INT_CLKO    |=  0x10;       //(EX2 = 1)使能INT2中断
    EA      =   1;

    while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机外部中断2 (INT2) (下降沿) ----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

INT_CLKO    DATA    08FH      //外部中断与时钟输出控制寄存器
//-----------------------------------------

            ORG     0000H
            LJMP    MAIN                        //复位入口

            ORG     0053H                       //INT2中断入口
            LJMP    EXINT2
//-----------------------------------------

            ORG     0100H
MAIN:
            MOV     SP,     #3FH

            ORL     INT_CLKO,       #10H        //(EX2 = 1)使能INT2中断

            SETB    EA

            SJMP    $
//-----------------------------------------
//外部中断服务程序

EXINT2:
            CPL     P1.0                        //将测试口取反

//          ANL     INT_CLKO,       #0EFH       //若需要手动清除中断标志,可先关闭中断,
                                                //此时系统会自动清除内部的中断标志
//          ORL     INT_CLKO,       #10H        //然后再开中断即可

            RETI
;-----------------------------------------
            END
```

STC MCU Limited

6.10.4 外部中断3($\overline{INT3}$)(下降沿中断)的测试程序(C和汇编)

1.C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机外部中断3 (INT3) (下降沿) ----------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------

sfr     INT_CLKO    =   0x8f;           //外部中断与时钟输出控制寄存器
sbit    P10     =   P1^0;

//-----------------------------------------------
//外部中断服务程序
void exint3() interrupt 11                      //INT3中断入口
{
    P10     =   !P10;                   //将测试口取反

//  INT_CLKO    &=  0xDF;               //若需要手动清除中断标志,可先关闭中断,
                                        //此时系统会自动清除内部的中断标志
//  INT_CLKO    |=  0x20;               //然后再开中断即可
}

void main()
{
    INT_CLKO    |=  0x20;               //(EX3 = 1)使能INT3中断
    EA      =   1;

    while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------*/
/* --- 演示STC 15 系列单片机外部中断3 (INT3) (下降沿) ---------------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com ------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

INT_CLKO    DATA    08FH                    //外部中断与时钟输出控制寄存器
//---------------------------------------

            ORG     0000H
            LJMP    MAIN                            //复位入口

            ORG     005BH                           //INT3中断入口
            LJMP    EXINT3
//---------------------------------------

            ORG     0100H
MAIN:
            MOV     SP,     #3FH

            ORL     INT_CLKO,       #20H            //(EX3 = 1)使能INT3中断

            SETB    EA

            SJMP    $

//---------------------------------------
//外部中断服务程序

EXINT3:
            CPL     P1.0                            //将测试口取反

//          ANL     INT_CLKO,       #0DFH           //若需要手动清除中断标志,可先关闭中断,
                                                    //此时系统会自动清除内部的中断标志
//          ORL     INT_CLKO,       #20H            //然后再开中断即可

            RETI
;---------------------------------------

            END
```

STC MCU Limited

6.10.5 外部中断4(INT4)(下降沿中断)的测试程序(C和汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机外部中断4 (INT4) (下降沿) ----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------

sfr     INT_CLKO    =   0x8f;       //外部中断与时钟输出控制寄存器
sbit    P10     =   P1^0;

//-----------------------------------------------
//外部中断服务程序
void exint4() interrupt 16                      //INT4中断入口
{
    P10     =   !P10;                           //将测试口取反

//  INT_CLKO    &=  0xBF;       //若需要手动清除中断标志,可先关闭中断,
                                //此时系统会自动清除内部的中断标志
//  INT_CLKO    |=  0x40;       //然后再开中断即可
}

void main()
{
    INT_CLKO    |=  0x40;       //(EX4 = 1)使能INT4中断
    EA      =   1;

    while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ---------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机外部中断4 (INT4) (下降沿) ----------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

INT_CLKO    DATA    08FH                        //外部中断与时钟输出控制寄存器

//-----------------------------------------
            ORG     0000H
            LJMP    MAIN                        //复位入口

            ORG     0083H                       //INT4中断入口
            LJMP    EXINT4
//-----------------------------------------

            ORG     0100H
MAIN:
            MOV     SP,     #3FH

            ORL     INT_CLKO,       #40H        //(EX4 = 1)使能INT4中断

            SETB    EA

            SJMP    $
//-----------------------------------------
//外部中断服务程序

EXINT4:
            CPL     P1.0                        //将测试口取反

//          ANL     INT_CLKO,       #0BFH       //若需要手动清除中断标志,可先关闭中断,
                                                //此时系统会自动清除内部的中断标志
//          ORL     INT_CLKO,       #40H        //然后再开中断即可

            RETI
;-----------------------------------------

            END
```

STC MCU Limited

6.10.6 T0扩展为外部下降沿中断的测试程序(C和汇编)

——利用T0的外部计数方式

1. C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T0扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------

sfr     AUXR    =   0x8e;              //辅助寄存器
sbit    P10     =   P1^0;

//-----------------------------------------------
//定时器0中断服务程序
void t0int() interrupt 1                //中断入口
{
        P10     =       !P10;           //将测试口取反
}

void main()
{
        AUXR    =       0x80;           //定时器0为1T模式
        TMOD    =       0x04;           //设置定时器0为16位自动重装载外部记数模式
        TH0 = TL0 =     0xff;           //设置定时器0初始值
        TR0     =       1;              //定时器0开始工作
        ET0     =       1;              //开定时器0中断

        EA      =       1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T0扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：  800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR    DATA  08EH                    //辅助寄存器
//-----------------------------------------

        ORG   0000H
        LJMP  MAIN                    //复位入口

        ORG   000BH                   //中断入口
        LJMP  T0INT
//-----------------------------------------

        ORG   0100H
MAIN:
        MOV   SP,   #3FH

        MOV   AUXR, #80H              //定时器0为1T模式
        MOV   TMOD, #04H              //设置定时器0为16位自动重装载外部记数模式
        MOV   A,    #0FFH             //设置定时器0初始值
        MOV   TL0,  A
        MOV   TH0,  A
        SETB  TR0                     //定时器0开始工作
        SETB  ET0                     //开定时器0中断

        SETB  EA

        SJMP  $

//-----------------------------------------
//定时器0中断服务程序

T0INT:
        CPL   P1.0                    //将测试口取反
        RETI
;-----------------------------------------

        END
```

STC MCU Limited

6.10.7 T1扩展为外部下降沿中断的测试程序(C和汇编)
——利用T1的外部计数方式

1. C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T1扩展为外部下降沿中断举例-----------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------

sfr     AUXR    =   0x8e;              //辅助寄存器
sbit    P10     =   P1^0;

//-----------------------------------------
//定时器1中断服务程序
void t1int() interrupt 3                //中断入口
{
        P10     =       !P10;           //将测试口取反
}

void main()
{
        AUXR    =       0x40;           //定时器1为1T模式
        TMOD    =       0x40;           //设置定时器1为16位自动重装载外部记数模式
        TH1 = TL1 =     0xff;           //设置定时器1初始值
        TR1     =       1;              //定时器1开始工作
        ET1     =       1;              //开定时器1中断

        EA      =        1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T1扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/


//假定测试芯片的工作频率为18.432MHz

AUXR    DATA  08EH                    //辅助寄存器
//-----------------------------------------

        ORG   0000H
        LJMP  MAIN                    //复位入口

        ORG   001BH                   //中断入口
        LJMP  T1INT
//-----------------------------------------

        ORG   0100H
MAIN:
        MOV   SP,   #3FH

        MOV   AUXR, #40H              //定时器1为1T模式
        MOV   TMOD, #40H              //设置定时器1为16位自动重装载外部记数模式
        MOV   A,    #0FFH             //设置定时器1初始值
        MOV   TL1,  A
        MOV   TH1,  A
        SETB  TR1                     //定时器1开始工作
        SETB  ET1                     //开定时器1中断

        SETB  EA

        SJMP  $

//-----------------------------------------
//定时器1中断服务程序

T1INT:
        CPL   P1.0                    //将测试口取反
        RETI
;-----------------------------------------

        END
```

STC MCU Limited

6.10.8 T2扩展为外部下降沿中断的测试程序(C和汇编)

——利用T2的外部计数方式

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T2扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------
sfr     IE2     =   0xaf;           //中断使能寄存器2
sfr     AUXR    =   0x8e;           //辅助寄存器
sfr     T2H     =   0xD6;           //定时器2高8位
sfr     T2L     =   0xD7;           //定时器2低8位

sbit P10 = P1^0;

//-----------------------------------------
//定时器2中断服务程序
void t2int() interrupt 12                   //中断入口
{
        P10     =   !P10;               //将测试口取反

//      IE2     &=  ~0x04;              //若需要手动清除中断标志,可先关闭中断,
                                        //此时系统会自动清除内部的中断标志
//      IE2     |=  0x04;               //然后再开中断即可
}

void main()
{
        AUXR    |=  0x04;               //定时器2为1T模式
```

STC MCU Limited

```
    AUXR        |= 0x08;                   //T2_C/T=1, T2(P3.1)引脚为时钟源
    T2H  = T2L = 0xff;                    //初始化计时值
    AUXR        |= 0x10;                  //定时器2开始计时

    IE2   |=    0x04;                     //开定时器2中断

    EA    =     1;

    while (1);
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T2扩展为外部下降沿中断举例-------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/


//假定测试芯片的工作频率为18.432MHz

IE2     DATA    0AFH                    //中断使能寄存器2
AUXR    DATA    08EH                    //辅助寄存器
T2H     DATA    0D6H                    //定时器2高8位
T2L     DATA    0D7H                    //定时器2低8位

//---------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     0063H                   //中断入口
        LJMP    T2INT
//---------------------------------------

        ORG     0100H
```

```
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #04H            //定时器2为1T模式
        ORL     AUXR,   #08H            //T2_C/T=1, T2(P3.1)引脚为时钟源

        MOV     A,      #0FFH           //初始化计时值
        MOV     T2L,    A
        MOV     T2H,    A

        ORL     AUXR,   #10H            //定时器2开始计时

        ORL     IE2,    #04H            //开定时器2中断

        SETB    EA

        SJMP    $
//---------------------------------------
//定时器2中断服务程序

T2INT:
        CPL     P1.0                    //将测试口取反

//      ANL     IE2,    #0FBH           //若需要手动清除中断标志,可先关闭中断,
                                        //此时系统会自动清除内部的中断标志
//      ORL     IE2,    #04H            //然后再开中断即可

        RETI

;---------------------------------------

        END
```

STC MCU Limited

6.10.9 用CCP/PCA功能扩展外部中断的测试程序(C和汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------*/
/* --- 演示  STC 1T 系列单片机  用PCA功能扩展外部中断 ---------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC    18432000L

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;
typedef   unsigned long     DWORD;

sfr       P_SW1   =   0xA2;                 //外设功能切换寄存器1

#define   CCP_S0  0x10                      //P_SW1.4
#define   CCP_S1  0x20                      //P_SW1.5

sfr       CCON    =   0xD8;                       //PCA控制寄存器
sbit      CCF0    =   CCON^0;                     //PCA模块0中断标志
sbit      CCF1    =   CCON^1;                     //PCA模块1中断标志
sbit      CR      =   CCON^6;                     //PCA定时器运行控制位
sbit      CF      =   CCON^7;                     //PCA定时器溢出标志
sfr       CMOD    =   0xD9;                       //PCA模式寄存器
sfr       CL      =   0xE9;                       //PCA定时器低字节
sfr       CH      =   0xF9;                       //PCA定时器高字节
sfr       CCAPM0      =   0xDA;                 //PCA模块0模式寄存器
sfr       CCAP0L      =   0xEA;                 //PCA模块0捕获寄存器 LOW
sfr       CCAP0H      =   0xFA;                 //PCA模块0捕获寄存器 HIGH
sfr       CCAPM1      =   0xDB;                 //PCA模块1模式寄存器
sfr       CCAP1L      =   0xEB;                 //PCA模块1捕获寄存器 LOW
sfr       CCAP1H      =   0xFB;                 //PCA模块1捕获寄存器 HIGH
sfr       CCAPM2      =   0xDC;                 //PCA模块2模式寄存器
sfr       CCAP2L      =   0xEC;                 //PCA模块2捕获寄存器 LOW
```

STC MCU Limited

```
sfr     CCAP2H          =       0xFC;           //PCA模块2捕获寄存器  HIGH

sfr     PCAPWM0         =       0xf2;           //PCA模块0的PWM寄存器
sfr     PCAPWM1         =       0xf3;           //PCA模块1的PWM寄存器
sfr     PCA_    PWM2    =       0xf4;           //PCA模块2的PWM寄存器

sbit    PCA_LED         =       P1^0;           //PCA测试LED

void PCA_isr() interrupt 7 using 1
{
        CCF0 = 0;                               //清中断标志
        PCA_LED = !PCA_LED;                     //测试LED取反
}

void main()
{
        ACC     =       P_SW1;
        ACC     &=      ~(CCP_S0 | CCP_S1);  //CCP_S0=0 CCP_S1=0
        P_SW1   =       ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      ACC     =       P_SW1;
//      ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=1 CCP_S1=0
//      ACC     |=      CCP_S0;         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      P_SW1   =       ACC;
//
//      ACC     =       P_SW1;
//      ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=1
//      ACC     |=      CCP_S1;         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      P_SW1   =       ACC;

        CCON    =        0;                     //初始化PCA控制寄存器
                                                //PCA定时器停止
                                                //清除CF标志
                                                //清除模块中断标志
        CL      =       0;                      //复位PCA寄存器
        CH      =       0;
        CMOD    =       0x00;                   //设置PCA时钟源
                                                //禁止PCA定时器溢出中断
        CCAPM0          =       0x11;           //PCA模块0为下降沿触发
//      CCAPM0          =       0x21;           //PCA模块0为上升沿沿触发
//      CCAPM0          =       0x31;           //PCA模块0为上升沿/下降沿触发

        CR      =       1;                      //PCA定时器开始工作
        EA      =       1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------*/
/* --- 演示  STC 1T 系列单片机  用PCA功能扩展外部中断 ----------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/


//假定测试芯片的工作频率为18.432MHz

//本测试程序以PCA模块0为例进行说明,PCA的模块1和模块2与模块0的实用方法相同

P_SW1   EQU     0A2H                    //外设功能切换寄存器1

CCP_S0  EQU     10H                     //P_SW1.4
CCP_S1  EQU     20H                     //P_SW1.5

CCON    EQU     0D8H                    ;PCA控制寄存器
CCF0    BIT     CCON.0                  ;PCA模块0中断标志
CCF1    BIT     CCON.1                  ;PCA模块1中断标志
CR      BIT     CCON.6                  ;PCA定时器运行控制位
CF      BIT     CCON.7                  ;PCA定时器溢出标志
CMOD    EQU     0D9H                    ;PCA模式寄存器
CL      EQU     0E9H                    ;PCA定时器低字节
CH      EQU     0F9H                    ;PCA定时器高字节
CCAPM0          EQU     0DAH            ;PCA模块0模式寄存器
CCAP0L          EQU     0EAH            ;PCA模块0捕获寄存器 LOW
CCAP0H          EQU     0FAH            ;PCA模块0捕获寄存器 HIGH
CCAPM1          EQU     0DBH            ;PCA模块1模式寄存器
CCAP1L          EQU     0EBH            ;PCA模块1捕获寄存器 LOW
CCAP1H          EQU     0FBH            ;PCA模块1捕获寄存器 HIGH
CCAPM2          EQU     0DCH            //PCA模块2模式寄存器
CCAP2L          EQU     0ECH            //PCA模块2捕获寄存器 LOW
CCAP2H          EQU     0FCH            //PCA模块2捕获寄存器 HIGH
PCA_PWM0        EQU     0F2H            //PCA模块0的PWM寄存器
PCA_PWM1        EQU     0F3H            //PCA模块1的PWM寄存器
PCA_PWM2        EQU     0F4H            //PCA模块2的PWM寄存器

PCA_LED BIT     P1.0                    ;PCA测试LED

;------------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     003BH
```

```
PCA_ISR:
        PUSH    PSW
        PUSH    ACC
CKECK_CCF0:
        JNB     CCF0,   PCA_ISR_EXIT    ;判断是否为捕获中断
        CLR     CCF0                    ;清中断标志
        CPL     PCA_LED                 ;测试LED取反
PCA_ISR_EXIT:
        POP     ACC
        POP     PSW
        RETI
;----------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #5FH

        MOV     A,      P_SW1
        ANL     A,      #0CFH           //CCP_S0=0 CCP_S1=0
        MOV     P_SW1,  A               //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      MOV     A,      P_SW1
//      ANL     A,      #0CFH           //CCP_S0=1 CCP_S1=0
//      ORL     A,      #CCP_S0         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      MOV     P_SW1,  A
//
//      MOV     A,      P_SW1
//      ANL     A,      #0CFH           //CCP_S0=0 CCP_S1=1
//      ORL     A,      #CCP_S1         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      MOV     P_SW1,  A

        MOV     CCON,   #0              ;初始化PCA控制寄存器
                                        ;PCA定时器停止
                                        ;清除CF标志
                                        ;清除模块中断标志
        CLR     A                       ;
        MOV     CL,     A               ;复位PCA寄存器
        MOV     CH,     A               ;
        MOV     CMOD,   #00H            ;设置PCA时钟源
                                        ;禁止PCA定时器溢出中断
        MOV     CCAPM0,   #11H          ;PCA模块0捕获CCP0(P1.3)的下降沿
;       MOV     CCAPM0,   #21H          ;PCA模块0捕获CCP0(P1.3)的上升沿
;       MOV     CCAPM0,   #31H          ;PCA模块0捕获CCP0(P1.3)的上升沿/下降沿
;-----------------------------
        SETB    CR                      ;PCA定时器开始工作
        SETB    EA

        SJMP    $
;----------------------------------------
        END
```

STC MCU Limited

第7章 定时器/计数器

STC15W4K60S4系列单片机内部设置了5个16位定时器/计数器：16位定时器/计数器T0和T1、T2、T3以及T4。5个16位定时器T0、T1、T2、T3和T4都具有计数方式和定时方式两种工作方式。对定时器/计数器T0和T1，用它们在特殊功能寄存器TMOD中相对应的控制位— C/$\overline{T}$来选择T0或T1为定时器还是计数器。对定时器/计数器T2，用特殊功能寄存器AUXR中的控制位— T2_C/$\overline{T}$来选择T2为定时器还是计数器。对定时器/计数器T3，用特殊功能寄存器T4T3M中的控制位— T3_C/$\overline{T}$来选择T3为定时器还是计数器。对定时器/计数器T4，用特殊功能寄存器T4T3M中的控制位— T4_C/$\overline{T}$来选择T4为定时器还是计数器。定时器/计数器的核心部件是一个加法计数器，其本质是对脉冲进行计数。只是计数脉冲来源不同：如果计数脉冲来自系统时钟，则为定时方式，此时定时器/计数器每12个时钟或者每1个时钟得到一个计数脉冲，计数值加1；如果计数脉冲来自单片机外部引脚(T0为P3.4，T1为P3.5，T2为P3.1，T3为P0.7，T4为P0.5)，则为计数方式，每来一个脉冲加1。

当定时器/计数器T0、T1及T2工作在定时模式时，特殊功能寄存器AUXR中的T0x12、T1x12和T2x12分别决定是系统时钟/12还是系统时钟/1(不分频)后让T0、T1和T2进行计数。当定时器/计数器T3和T4工作在定时模式时，特殊功能寄存器T4T3M中的T3x12和T4x12分别决定是系统时钟/12还是系统时钟/1(不分频)后让T3和T4进行计数。当定时器/计数器工作在计数模式时，对外部脉冲计数不分频。

定时器/计数器0有4种工作模式：模式0(16位自动重装载模式)，模式1(16位不可重装载模式)，模式2(8位自动重装模式)，模式3(不可屏蔽中断的16位自动重装载模式)。定时器/计数器1除模式3外，其他工作模式与定时器/计数器0相同，T1在模式3时无效，停止计数。定时器T2的工作模式固定为16位自动重装载模式。T2可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。定时器3、定时器4与定时器T2一样，它们的工作模式固定为16位自动重装载模式。T3/T4可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。

STC15全系列的定时器/计数器的类型如下表所示。

定时器/计数器 / 单片机型号	定时器/计数器0	定时器/计数器1	定时器/计数器2	定时器/计数器3	定时器/计数器4
STC15F101W系列	√		√		
STC15F408AD系列	√		√		
STC15W201S系列	√		√		
STC15W408AS系列	√		√		
STC15W408S系列	√	√	√		
STC15W1K16S系列	√	√	√		
STC15F2K60S2系列	√	√	√		
STC15W4K60S4系列	√	√	√	√	√

上表中√表示对应的系列有相应的定时器/计数器。

7.1 定时器/计数器的相关寄存器

符号	描述	地址	位地址及其符号 MSB LSB								复位值
TCON	Timer Control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
TMOD	Timer Mode	89H	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0	0000 0000B
TL0	Timer Low 0	8AH									0000 0000B
TL1	Timer Low 1	8BH									0000 0000B
TH0	Timer High 0	8CH									0000 0000B
TH1	Timer High 1	8DH									0000 0000B
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	中断优先级寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
T2H	定时器2高8位寄存器	D6H									0000 0000B
T2L	定时器2低8位寄存器	D7H									0000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	外部中断允许和时钟输出寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000 x000B
T4T3M	T4和T3的控制寄存器	D1H	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO	0000 0000B
T4H	定时器4高8位寄存器	D2H									0000 0000B
T4L	定时器4低8位寄存器	D3H									0000 0000B
T3H	定时器3高8位寄存器	D4H									0000 0000B
T3L	定时器3低8位寄存器	D5H									0000 0000B
IE2	Interrupt Enable register	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B

STC MCU Limited

1. 定时器/计数器0/1控制寄存器TCON

TCON为定时器/计数器T0、T1的控制寄存器，同时也锁存T0、T1溢出中断源和外部请求中断源等，TCON格式如下：

TCON：定时器/计数器中断控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	name	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：T1溢出中断标志。T1被允许计数以后，从初值开始加1计数。当产生溢出时由硬件置“1”TF1，向CPU请求中断，一直保持到CPU响应中断时，才由硬件清“0”（也可由查询软件清“0”）。

TR1：定时器T1的运行控制位。该位由软件置位和清零。当GATE（TMOD.7）=0，TR1=1时就允许T1开始计数，TR1=0时禁止T1计数。当GATE（TMOD.7）=1，TR1=1且INT1输入高电平时，才允许T1计数。

TF0：T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

TR0：定时器T0的运行控制位。该位由软件置位和清零。当GATE（TMOD.3）=0，TR0=1时就允许T0开始计数，TR0=0时禁止T0计数。当GATE（TMOD.3）=1，TR1=1且INT0输入高电平时，才允许T0计数，TR0=0时禁止T0计数。

IE1：外部中断1请求源（INT1/P3.3）标志。IE1=1，外部中断向CPU请求中断，当CPU响应该中断时由硬件清“0”IE1。

IT1：外部中断源1触发控制位。IT1=0，上升沿或下降沿均可触发外部中断1。IT1=1，外部中断1程控为下降沿触发方式。

IE0：外部中断0请求源（INT0/P3.2）标志。IE0=1外部中断0向CPU请求中断，当CPU响应外部中断时，由硬件清“0”IE0（边沿触发方式）。

IT0：外部中断源0触发控制位。IT0=0，上升沿或下降沿均可触发外部中断0。IT0=1，外部中断0程控为下降沿触发方式。

STC MCU Limited

2. 定时器/计数器工作模式寄存器TMOD

定时和计数功能由特殊功能寄存器TMOD的控制位C/$\overline{T}$进行选择，TMOD寄存器的各位信息如下表所列。可以看出，2个定时/计数器有4种操作模式，通过TMOD的M1和M0选择。2个定时/计数器的模式0、1和2都相同，模式3不同，各模式下的功能如下所述。

寄存器TMOD各位的功能描述

TMOD　　地址：89H　　复位值：00H

不可位寻址

7	6	5	4	3	2	1	0
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
定时器1				定时器0			

位	符号	功能
TMOD.7/	GATE	TMOD.7控制定时器1,置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。
TMOD.3/	GATE	TMOD.3控制定时器0,置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。
TMOD.6/	C/$\overline{T}$	TMOD.6控制定时器1用作定时器或计数器，清零则用作定时器(对内部系统时钟进行计数)，置1用作计数器(对引脚T1/P3.5外部脉冲进行计数)
TMOD.2/	C/$\overline{T}$	TMOD.2控制定时器0用作定时器或计数器，清零则用作定时器(对内部系统时钟进行计数)，置1用作计数器(对引脚T0/P3.4的外部脉冲进行计数)
TMOD.5/TMOD.4	M1、M0	定时器定时器/计数器1模式选择
	0　0	16位自动重装定时器，当溢出时将RL_TH1和RL_TL1存放的值自动重装入TH1和TL1中。
	0　1	16位不可重装载模式，TL1、TH1全用
	1　0	8位自动重装载定时器,当溢出时将TH1存放的值自动重装入TL1
	1　1	定时器/计数器1此时无效（停止计数）。
TMOD.1/TMOD.0	M1、M0	定时器/计数器0模式选择
	0　0	16位自动重装定时器，当溢出时将RL_TH0和RL_TL0存放的值自动重装入TH0和TL0中。
	0　1	16位不可重装载模式，TL0、TH0全用
	1　0	8位自动重装载定时器，当溢出时将TH0存放的值自动重装入TL0
	1　1	不可屏蔽中断的16位自动重装定时器

STC MCU Limited

3. 辅助寄存器AUXR

STC15系列单片机是 1T 的8051单片机，为兼容传统8051，定时器0、定时器1,和定时器2复位后是传统8051的速度，即12分频，这是为了兼容传统8051。但也可不进行12分频，通过设置新增加的特殊功能寄存器AUXR,将T0,T1,T2设置为1T。普通111条机器指令执行速度是固定的，快4到24倍，无法改变。

AUXR格式如下：

AUXR：辅助寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2

T0x12：定时器0速度控制位

0，定时器0是传统8051速度，12分频；

1，定时器0的速度是传统8051的12倍，不分频

T1x12：定时器1速度控制位

0，定时器1是传统8051速度，12分频；

1，定时器1的速度是传统8051的12倍，不分频

如果UART1/串口1用T1作为波特率发生器，则由T1x12决定UART1/串口1是12T还是1T

UART_M0x6：串口1模式0的通信速度设置位。

0，串口1模式0的速度是传统8051单片机串口的速度，12分频；

1，串口1模式0的速度是传统8051单片机串口速度的6倍，2分频

T2R：定时器2允许控制位

0，不允许定时器2运行；

1，允许定时器2运行

T2_C/$\overline{T}$：控制定时器2用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T2/P3.1的外部脉冲进行计数)

T2x12：定时器2速度控制位

0，定时器2是传统8051速度，12分频；

1，定时器2的速度是传统8051的12倍，不分频

如果串口1或串口2用T2作为波特率发生器，则由T2x12决定串口1或串口2是12T还是1T.

EXTRAM：内部/外部RAM存取控制位

0，允许使用逻辑上在片外、物理上在片内的扩展RAM；

1，禁止使用逻辑上在片外、物理上在片内的扩展RAM

S1ST2：串口1(UART1)选择定时器2作波特率发生器的控制位

0，选择定时器1作为串口1(UART1)的波特率发生器；

1，选择定时器2作为串口1(UART1)的波特率发生器，此时定时器1得到释放，可以作为独立定时器使用

STC MCU Limited

4. T0, T1和T2的时钟输出寄存器和外部中断允许INT_CLKO (AUXR2)

T0CLKO/P3.5、T1CLKO/P3.4和T2CLKO/P3.0的时钟输出控制由INT_CLKO(AUXR2)寄存器的T0CLKO位、T1CLKO位和T2CLKO位控制。T0CLKO的输出时钟频率由定时器0控制, T1CLKO的输出时钟频率由定时器1控制, 相应的定时器需要工作在定时器的模式0(16位自动重装载模式)或模式2(8位自动重装载模式), 不要允许相应的定时器中断, 免得CPU反复进中断。T2CLKO的输出时钟频率由定时器2控制, 同样不要允许相应的定时器中断, 免得CPU反复进中断。定时器2的工作模式固定为模式0(16位自动重装载模式), 在此模式下定时器2可用作可编程时钟输出。

INT_CLKO (AUXR2)格式如下：

INT_CLKO (AUXR2): 外部中断允许和时钟输出寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T0CLKO ： 是否允许将P3.5/T1脚配置为定时器0(T0)的时钟输出T0CLKO

1，将P3.5/T1管脚配置为定时器0的时钟输出T0CLKO，输出时钟频率=T0溢出率/2

若定时器/计数器T0工作在定时器模式0(16位自动重装载模式)时，

如果C/$\overline{T}$=0，定时器/计数器T0是对内部系统时钟计数，则：

T0工作在1T模式(AUXR.7/T0x12=1)时的输出频率 = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH0, RL_TL0])/2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

若定时器/计数器T0工作在定时器模式2(8位自动重装模式)，

如果C/$\overline{T}$=0, 定时器/计数器T1是对内部系统时钟计数，则：

T0工作在1T模式(AUXR.7/T0x12=1)时的输出频率 = (SYSclk) / (256-TH0) / 2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出频率 = (SYSclk) / 12 / (256-TH0) / 2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (256-TH0) / 2

0，不允许P3.5/T1管脚被配置为定时器0的时钟输出

T1CLKO：是否允许将P3.4/T0脚配置为定时器1(T1)的时钟输出T1CLKO

1，将P3.4/T0管脚配置为定时器1的时钟输出T1CLKO，输出时钟频率= T1溢出率/2

若定时器/计数器T1工作在定时器模式0(16位自动重装载模式)，

如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH1, RL_TL1])/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH1, RL_TL1])/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

若定时器/计数器T1工作在模式2(8位自动重装模式)，

如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (256-TH1)/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk)/12/(256-TH1)/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1) / 2

0，不允许P3.4/T0管脚被配置为定时器1的时钟输出

T2CLKO：是否允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

1：允许将P3.0脚配置为定时器2的时钟输出T2CLKO，输出时钟频率=T2溢出率/2

如果T2_C/$\overline{T}$=0，定时器/计数器T2是对内部系统时钟计数，则：

T2工作在1T模式(AUXR.2/T2x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH2, RL_TL2])/2

T2工作在12T模式(AUXR.2/T2x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH2, RL_TL2])/2

如果T2_C/$\overline{T}$=1，定时器/计数器T2是对外部脉冲输入(P3.1/T2)计数，则：

输出时钟频率 = (T2_Pin_CLK) / (65536-[RL_TH2, RL_TL2])/2

0： 不允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

EX4： 外部中断4($\overline{INT4}$)中断允许位，EX4=1允许中断，EX4=0禁止中断。外部中断4($\overline{INT4}$)只能下降沿触发。

EX3： 外部中断3($\overline{INT3}$)中断允许位，EX3=1允许中断，EX3=0禁止中断。外部中断3($\overline{INT3}$)也只能下降沿触发。

EX2： 外部中断2($\overline{INT2}$)中断允许位，EX2=1允许中断，EX2=0禁止中断。外部中断2($\overline{INT2}$)同样只能下降沿触发。

STC MCU Limited

5、定时器T0和T1的中断控制寄存器:IE和IP

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的总中断允许控制位，EA=1，CPU开放中断，EA=0，CPU屏蔽所有的中断申请。
EA的作用是使中断允许形成多级控制。即各中断源首先受EA控制;其次还受各中断源自己的中断允许控制位控制。

ET1：定时/计数器T1的溢出中断允许位，ET1=1，允许T1中断，ET1=0，禁止T1中断。

ET0：T0的溢出中断允许位，ET0=1允许T0中断，ET0=0禁止T0中断。

IP：中断优先级控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PT1:　定时器1中断优先级控制位。
当PT1=0时，定时器1中断为最低优先级中断(优先级0)
当PT1=1时，定时器1中断为最高优先级中断(优先级1)

PT0:　定时器0中断优先级控制位。
当PT0=0时，定时器0中断为最低优先级中断(优先级0)
当PT0=1时，定时器0中断为最高优先级中断(优先级1)

注意：当定时器/计数器0工作在模式3(不可屏蔽中断的16位自动重装载模式)时，不需要允许EA/IE.7(总中断使能位)只需允许ET0/IE.1(定时器/计数器0中断允许位)就能打开定时器/计数器0的中断，此模式下的定时器/计数器0中断与总中断使能位EA无关。一旦此模式下的定时器/计数器0中断被打开后，该定时器/计数器0中断优先级就是最高的，它不能被其它任何中断所打断(不管是比定时器/计数器0中断优先级低的中断还是比其优先级高的中断，都不能打断此时的定时器/计数器0中断)，而且该中断打开后既不受EA/IE.7控制也不再受ET0控制了，清零EA或ET0都不能关闭此中断。

STC MCU Limited

6、定时器T4和T3的控制寄存器：T4T3M（地址：0xD1）

T4T3M（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO

B7 - T4R：定时器4运行控制位。
　　0：不允许定时器4运行；
　　1：允许定时器4运行。

B6 - T4_C/$\overline{T}$：控制定时器4用作定时器或计数器。
　　0，用作定时器（对内部系统时钟进行计数）；
　　1，用作计数器（对引脚T4/P0.7的外部脉冲进行计数）

B5 - T4x12：定时器4速度控制位。
　　0：定时器4速度是8051单片机定时器的速度，即12分频；
　　1：定时器4速度是8051单片机定时器速度的12倍，即不分频。

B4 - T4CLKO：是否允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO
　1：允许将P0.6脚配置为定时器4的时钟输出T4CLKO，输出时钟频率=T4溢出率/2
　　如果T4_C/$\overline{T}$=0，定时器/计数器T4是对内部系统时钟计数，则：
　　T4工作在1T模式(T4T3M.5/T4x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH4, RL_TL4])/2
　　T4工作在12T模式(T4T3M.5/T4x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH4, RL_TL4])/2
　　如果T4_C/$\overline{T}$=1，定时器/计数器T4是对外部脉冲输入(P0.7/T4)计数，则：
　　输出时钟频率 = (T4_Pin_CLK) / (65536-[RL_TH4, RL_TL4])/2
　0：不允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

B3 - T3R：定时器3运行控制位。
　　0：不允许定时器3运行；
　　1：允许定时器3运行。

B2 - T3_C/$\overline{T}$：控制定时器3用作定时器或计数器。
　　0，用作定时器（对内部系统时钟进行计数）；
　　1，用作计数器（对引脚T3/P0.5的外部脉冲进行计数）

B1 - T3x12：定时器3速度控制位。
　　0：定时器3速度是8051单片机定时器的速度，即12分频；
　　1：定时器3速度是8051单片机定时器速度的12倍，即不分频。

B0 - T3CLKO：是否允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO
　1：允许将P0.4脚配置为定时器3的时钟输出T3CLKO，输出时钟频率=T3溢出率/2
　　如果T3_C/$\overline{T}$=0，定时器/计数器T3是对内部系统时钟计数，则：
　　T3工作在1T模式(T4T3M.1/T3x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH3, RL_TL3])/2
　　T3工作在12T模式(T4T3M.1/T3x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH3, RL_TL3])/2
　　如果T3_C/$\overline{T}$=1，定时器/计数器T3是对外部脉冲输入(P0.5/T3)计数，则：
　　输出时钟频率 = (T3_Pin_CLK) / (65536-[RL_TH3, RL_TL3])/2
　0：不允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

STC-ISP Limited

7、定时器T2、T3和T4的中断控制寄存器:IE2

IE2：中断允许寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4：定时器4的中断允许位。

1，允许定时器4产生中断；

0，禁止定时器4产生中断。

ET3：定时器3的中断允许位。

1，允许定时器3产生中断；

0，禁止定时器3产生中断。

ES4：串行口4中断允许位。

1，允许串行口4中断；

0，禁止串行口4中断

ES3：串行口3中断允许位。

1，允许串行口3中断；

0，禁止串行口3中断。

ET2：定时器2的中断允许位。

1，允许定时器2产生中断；

0，禁止定时器2产生中断。

ESPI：SPI中断允许位。

1，允许SPI中断；

0，禁止SPI中断。

ES2：串行口2中断允许位。

1，允许串行口2中断；

0，禁止串行口2中断。

STC MCU Limited

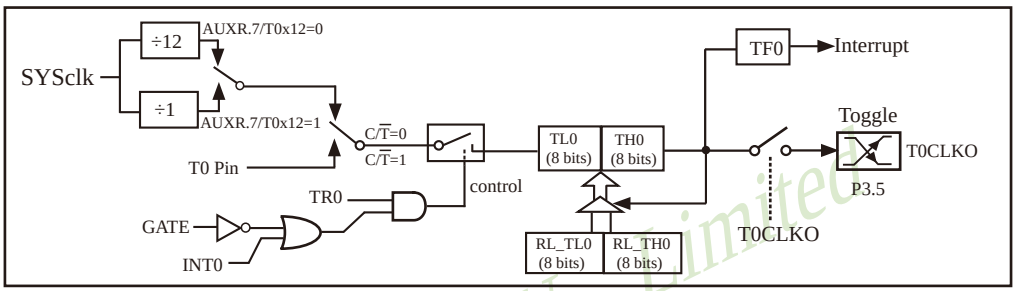
7.2 定时器/计数器0工作模式

通过对寄存器TMOD中的M1(TMOD.1)、M0(TMOD.0)的设置，定时器/计数器0有4种不同的工作模式

7.2.1 模式0(16位自动重装载模式)及测试程序，建议只学习此模式足矣

——STC创新设计，请不要抄袭

此模式下定时器/计数器0作为可自动重装载的16位计数器，如下图所示。



定时器/计数器0的模式 0: 16位自动重装

STC创新设计，请不要抄袭，再抄袭就很无耻了

当GATE=0(TMOD.3)时，如TR0=1，则定时器计数。GATE=1时，允许由外部输入INT0控制定时器0，这样可实现脉宽测量。TR0为TCON 寄存器内的控制位，TCON寄存器各位的具体功能描述见上节TCON寄存器的介绍。

当C/$\overline{T}$=0时，多路开关连接到系统时钟的分频输出，T0对内部系统时钟计数，T0工作在定时方式。当C/$\overline{T}$=1时，多路开关连接到外部脉冲输入P3.4/T0，即T0工作在计数方式。

STC15系列单片机的定时器有两种计数速率：一种是12T模式，每12个时钟加1，与传统8051单片机相同；另外一种是1T模式，每个时钟加1，速度是传统8051单片机的12倍。T0的速率由特殊功能寄存器AUXR中的T0x12决定，如果T0x12=0，T0则工作在12T模式；如果T0x12=1，T0则工作在1T模式。

定时器0有2个隐藏的寄存器RL_TH0和RL_TL0。RL_TH0与TH0共有同一个地址，RL_TL0与TL0共有同一个地址。当TR0=0即定时器/计数器0被禁止工作时，对TL0写入的内容会同时写入RL_TL0，对TH0写入的内容也会同时写入RL_TH0。当TR0=1即定时器/计数器0被允许工作时，对TL0写入内容，实际上不是写入当前寄存器TL0中,而是写入隐藏的寄存器RL_TL0中；对TH0写入内容，实际上也不是写入当前寄存器TH0中，而是写入隐藏的寄存器RL_TH0。这样可以巧妙地实现16位重装载定时器。当读TH0和TL0的内容时，所读的内容就是TH0和TL0的内容，而不是RL_TH0和RL_TL0的内容。

当定时器0工作在模式0(TMOD[1:0]/[M1,M0]=00B)时，[TL0, TH0]的溢出不仅置位TF0，而且会自动将[RL_TL0,RL_TH0]的内容重新装入[TL0,TH0]。

当T0CLKO/INT_CLKO.0=1时，P3.5/T1管脚配置为定时器0的时钟输出T0CLKO。
输出时钟频率 = T0 溢出率/2
　　如果C/$\overline{T}$=0，定时器/计数器T0对内部系统时钟计数，则：
　　T0工作在1T模式(AUXR.7/T0x12=1)时的输出**时钟频率** = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2
　　T0工作在12T模式(AUXR.7/T0x12=0)时的输出**时钟频率**=(SYSclk)/12/(65536-[RL_TH0, RL_TL0])/2
　　如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：
　　输出**时钟频率** = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

7.2.1.1 定时器0的16位自动重装载模式的测试程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------------------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机定时器0的16位自动重装载模式 ------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com ------------------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ---------------------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef unsigned char       BYTE;
typedef unsigned int        WORD;

//-----------------------------------------------

#define   FOSC    18432000L

#define   T1MS    (65536-FOSC/1000)             //1T模式，18.432MHz
//#define T1MS    (65536-FOSC/12/1000)          //12T模式，18.432MHz

sfr       AUXR    =       0x8e;                 //Auxiliary register
sbit      P10     =       P1^0;

//-----------------------------------------------
```

```
/* Timer0 interrupt routine */
void tm0_isr() interrupt 1 using 1
{
        P10     =       ! P10;                  //将测试口取反
}

//-----------------------------------------

/* main program */
void main()
{
        AUXR    |=      0x80;                   //定时器0为1T模式
//      AUXR    &=      0x7f;                   //定时器0为12T模式

        TMOD    =       0x00;                   //设置定时器为模式0(16位自动重装载)
        TL0     =       T1MS;                   //初始化计时值
        TH0     =       T1MS >> 8;
        TR0     =       1;                      //定时器0开始计时
        ET0     =       1;                      //使能定时器0中断
        EA      =       1;

        while (1);
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ---------------------------------------------------*/
/* --- 演示STC 15 系列单片机定时器0的16位自动重装载模式 ----------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ---------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR    DATA    08EH                    //辅助特殊功能寄存器

;-----------------------------------------

T1MS    EQU     0B800H                  //1T模式的1ms定时值(65536-18432000/1000)
//T1MS  EQU     0FA00H                  //12T模式的1ms定时值(65536-18432000/1000/12)

;-----------------------------------------
```

STC MCU Limited

```
        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     000BH                   //中断入口
        LJMP    T0INT

;-------------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #80H            //定时器0为1T模式
//      ANL     AUXR,   #7FH            //定时器0为12T模式

        MOV     TMOD,   #00H            //设置定时器为模式0(16位自动重装载)

        MOV     TL0,    #LOW T1MS       //初始化计时值
        MOV     TH0,    #HIGH T1MS
        SETB    TR0
        SETB    ET0                     //使能定时器0中断

        SETB    EA

        SJMP    $                       //程序终止

//-------------------------------------------
//中断服务程序

T0INT:
        CPL     P1.0                    //将测试口取反
        RETI

;-------------------------------------------

        END
```

STC MCU Limited

7.2.1.2 定时器0对系统时钟或外部引脚T0的时钟输入进行可编程分频输出的测试程序——定时器0工作在16位自动重装载模式

下面是定时器0工作在16位重装模式时对内部系统时钟或外部引脚T0/P3.4的时钟输入进行可编程时钟分频输出的程序举例(C和汇编)：

1. C程序：

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------------*/
/* --- 演示STC15F2K60S2 系列单片机定时器0的可编程时钟分频输出-----*/
/* --- 研发顾问QQ：800003751-------------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------------*/
/* --- Tel: 86-755-82948412 --------------------------------------------------------*/
/* --- Web: www.STCMCU.com ---------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define FOSC 18432000L

//-----------------------------------------------
sfr     AUXR        =     0x8e;              //辅助特殊功能寄存器
sfr     INT_CLKO    =     0x8f;              //唤醒和时钟输出功能寄存器

sbit    T0CLKO      =     P3^5;              //定时器0的时钟输出脚

#define F38_4KHz    (65536-FOSC/2/38400)     //1T模式
//#define F38_4KHz  (65536-FOSC/2/12/38400)  //12T模式

//-----------------------------------------------

void main()
{
        AUXR    |=    0x80;                  //定时器0为1T模式
//      AUXR    &=    ~0x80;                 //定时器0为12T模式
```

STC MCU Limited

```
        TMOD    =       0x00;                   //设置定时器为模式0(16位自动重装载)

        TMOD    &=      ~0x04;                  //C/T0=0, 对内部时钟进行时钟输出
//      TMOD    |=      0x04;                   //C/T0=1, 对T0引脚的外部时钟进行时钟输出

        TL0     =       F38_4KHz;               //初始化计时值
        TH0     =       F38_4KHz >> 8;
        TR0     =       1;
        INT_CLKO        =       0x01;           //使能定时器0的时钟输出功能

        while (1);                              //程序终止
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------------*/
/* --- 演示STC15F2K60S2 系列单片机定时器0的可编程时钟分频输出-----*/
/* --- 研发顾问QQ：800003751-------------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------------*/
/* --- Tel: 86-755-82948412 --------------------------------------------------------*/
/* --- Web: www.STCMCU.com ---------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR            DATA    08EH            //辅助特殊功能寄存器
INT_CLKO        DATA    08FH            //唤醒和时钟输出功能寄存器

T0CLKO          BIT     P3.5            //定时器0的时钟输出脚

F38_4KHz        EQU     0FF10H          //38.4KHz(1T模式下, 65536-18432000/2/38400)
//F38_4KHz      EQU     0FFECH          //38.4KHz(12T模式下, (65536-18432000/2/12/38400)

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口
```

STC MCU Limited

```
//---------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,         #3FH

        ORL     AUXR,   #80H                    //定时器0为1T模式
//      ANL     AUXR,   #7FH                    //定时器0为12T模式

        MOV     TMOD,   #00H                    //设置定时器为模式0(16位自动重装载)

        ANL     TMOD,   #0FBH                   //C/T0=0, 对内部时钟进行时钟输出
//      ORL     TMOD,   #04H                    //C/T0=1, 对T0引脚的外部时钟进行时钟输出

        MOV     TL0,        #LOW F38_4KHz       //初始化计时值
        MOV     TH0,        #HIGH F38_4KHz
        SETB    TR0
        MOV     INT_CLKO,           #01H        //使能定时器0的时钟输出功能

        SJMP    $                               //程序终止

;---------------------------------------------

        END
```

STC MCU Limited

7.2.1.3 T0的16位自动重装模式(软硬结合)模拟10位或16位PWM输出的程序(C和汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器软件模拟PWM举例------------------------------------------------------ */
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

//#define  PWM6BIT        64                      //6-bit PWM 周期数
#define   PWM8BIT        256                     //8-bit PWM 周期数
//#define  PWM10BIT       1024                    //10-bit PWM 周期数
//#define  PWM16BIT       65536                   //16-bit PWM 周期数

#define   HIGHDUTY       64                      //高电平周期数(占空比64/256=25%)
#define   LOWDUTY        (PWM8BIT-HIGHDUTY)      //低电平周期数

sfr       AUXR           =       0x8e;           //辅助寄存器
sfr       INT_CLKO       =       0x8f;           //时钟输出控制寄存器
sbit      T0CLKO         =       P3^5;           //定时器0的时钟输出口

bit       flag;

//定时器0中断服务程序
void tm0() interrupt 1
{
        flag = !flag;                             //反转PWM的输出标志
        if (flag)
        {
                TL0 = (65536-HIGHDUTY);           //准备高电平的重载值
                TH0 = (65536-HIGHDUTY) >> 8;
        }
        else
        {
                TL0 = (65536-LOWDUTY);            //准备低电平的重载值
                TH0 = (65536-LOWDUTY) >> 8;
        }
}
```

STC MCU Limited

```
void main()
{
    AUXR    =   0x80;                   //定时器0为1T模式
    INT_CLKO    =   0x01;               //使能定时器0的时钟输出功能
    TMOD    &=  0xf0;                   //设置定时器0为模式0(16位自动重装载)
    TL0     =   (65536-LOWDUTY);        //初始化定时器初值和重装值
    TH0     =   (65536-LOWDUTY) >> 8;
    T0CLKO  =   1;                      //初始化时钟输出脚(软PWM口)
    flag    =   0;                      //初始化标志位
    TR0     =   1;                      //定时器0开始计时
    ET0     =   1;                      //使能定时器0中断
    EA      =   1;
    while (1);
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ------------------------------------------------------------------------------ */
/* --- STC15F2K60S2 系列 定时器软件模拟PWM举例-------------------------------------------------------- */
/* --- 研发顾问QQ：  800003751------------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 --------------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ---------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                    */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                    */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

;PWM6BIT        EQU     64                  ;6-bit PWM周期数
PWM8BIT         EQU     256                 ;8-bit PWM周期数
;PWM10BIT       EQU     1024                ;10-bit PWM周期数
;PWM16BIT       EQU     65536               ;16-bit PWM周期数

HIGHDUTY        EQU     64                  ;高电平周期数(占空比64/256=25%)
LOWDUTY         EQU     (PWM8BIT-HIGHDUTY)  ;低电平周期数

AUXR            DATA    08EH                ;辅助寄存器
INT_CLKO        DATA    08FH                ;时钟输出控制寄存器
T0CLKO          BIT     P3.5                ;定时器0的时钟输出口

FLAG            BIT     20H.0

;-----------------------------------------
```

STC MCU Limited

```
        ORG     0000H
        LJMP    MAIN

        ORG     000BH
        LJMP    TM0_ISR

;---------------------------------------------

MAIN:
        MOV     AUXR,   #80H                    ;定时器0为1T模式
        MOV     INT_CLKO,       #01H            ;使能定时器0的时钟输出功能
        ANL     TMOD,   #0F0H                   ;设置定时器0为模式0(16位自动重装载)
        MOV     TL0,    #LOW (65536-LOWDUTY)    ;初始化定时器初值和重装值
        MOV     TH0,    #HIGH (65536-LOWDUTY)
        SETB    T0CLKO                          ;初始化时钟输出脚(软PWM口)
        CLR     FLAG                            ;初始化标志位
        SETB    TR0                             ;定时器0开始计时
        SETB    ET0                             ;使能定时器0中断
        SETB    EA

        SJMP    $

;---------------------------------------------
;定时器0中断服务程序
TM0_ISR:
        CPL     FLAG                            ;反转PWM的输出标志
        JNB     FLAG,   READYLOW
READYHIGH:
        MOV     TL0,    #LOW (65536-HIGHDUTY)   ;准备高电平的重载值
        MOV     TH0,    #HIGH (65536-HIGHDUTY)
        JMP     TM0ISR_EXIT
READYLOW:
        MOV     TL0,    #LOW (65536-LOWDUTY)    ;准备低电平的重载值
        MOV     TH0,    #HIGH (65536-LOWDUTY)
TM0ISR_EXIT:
        RETI

;---------------------------------------------

        END
```

STC MCU Limited

7.2.1.4 T0的16位自动重装载模式扩展为外部下降沿中断的测试程序(C和汇编)
——利用T0的外部计数方式

;定时器0中断(下降沿中断)的测试程序，定时器/计数器0工作在计数模式中的16位自动重装载模式，定时器/计数器0工作载外部计数模式。

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T0扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------
sfr     AUXR    =       0x8e;                   //辅助寄存器
sbit    P10     =       P1^0;

//-----------------------------------------------
//外部中断服务程序
void t0int() interrupt 1                        //中断入口
{
        P10     =       !P10;                   //将测试口取反
}

void main()
{
        AUXR    =       0x80;                   //定时器0为1T模式
        TMOD    =       0x04;   //设置定时器0工作在16位自动重装载模式，同时为外部记数模式
        TH0     =       0xff;
        TL0     =       0xff;                   //设置定时器0初始值
        TR0     =       1;                      //定时器0开始工作
        ET0     =       1;                      //开定时器0中断

        EA      =       1;

        while (1);
}
```

STC MCU Limited

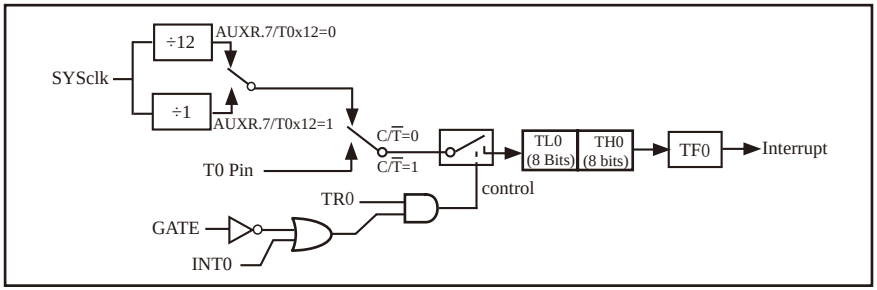
2. 汇编程序：

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T0扩展为外部下降沿中断举例------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR    DATA   08EH                    //辅助寄存器
//-----------------------------------------

        ORG    0000H
        LJMP   MAIN                    //复位入口

        ORG    000BH                   //中断入口
        LJMP   T0INT
//-----------------------------------------

        ORG    0100H
MAIN:
        MOV    SP,    #3FH

        MOV    AUXR,  #80H             //定时器0为1T模式
        MOV    TMOD,  #04H             //设置定时器0工作在16位自动重装载模式，
                                       //同时为为外部记数模式
        MOV    A,     #0FFH            //设置定时器0初始值
        MOV    TL0,   A
        MOV    TH0,   A
        SETB   TR0                     //定时器0开始工作
        SETB   ET0                     //开定时器0中断

        SETB   EA

        SJMP   $

//-----------------------------------------
//外部中断服务程序

T0INT:
        CPL    P1.0                    //将测试口取反
        RETI
;-----------------------------------------

        END
```

STC MCU Limited

7.2.2 模式1(16位不可重装载模式)，不建议学习

此模式下定时器/计数器0工作在16位不可重装载模式，如下图所示。



定时器/计数器0的模式 1: 16位不可重装载模式

此模式下，定时器/计数器0配置为16位不可重装载模式，由TL0的8位和TH0的8位所构成。TL0的8位溢出向TH0进位，TH0计数溢出置位TCON中的溢出标志位TF0。

当GATE=0(TMOD.3)时，如TR0=1，则定时器计数。GATE=1时，允许由外部输入INT0控制定时器0，这样可实现脉宽测量。TR0为TCON寄存器内的控制位，TCON寄存器各位的具体功能描述见上节TCON寄存器的介绍。

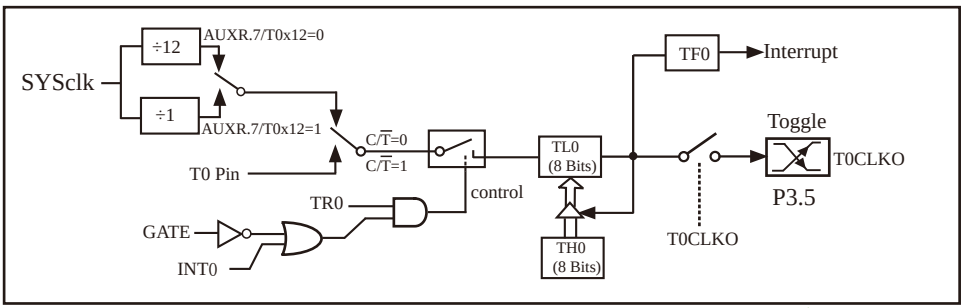
当C/$\overline{T}$=0时，多路开关连接到系统时钟的分频输出，T0对内部系统时钟计数，T0工作在定时方式。当C/$\overline{T}$=1时，多路开关连接到外部脉冲输入P3.4/T0，即T0工作在计数方式。

STC15系列单片机的定时器有两种计数速率：一种是12T模式，每12个时钟加1，与传统8051单片机相同；另外一种是1T模式，每个时钟加1，速度是传统8051单片机的12倍。T0的速率由特殊功能寄存器AUXR中的T0x12决定，如果T0x12=0，T0则工作在12T模式；如果T0x12=1，T0则工作在1T模式。

STC MCU Limited

7.2.3 模式2(8位自动重装载模式)，不建议学习

此模式下定时器/计数器0作为可自动重装载的8位计数器，如下图所示。



定时器/计数器0的模式 2: 8位自动重装

TL0的溢出不仅置位TF0，而且将TH0内容重新装入TL0，TH0内容由软件预置，重装时TH0内容不变。

当T0CLKO/INT_CLKO.0=1时，P3.5/T1管脚配置为定时器0的时钟输出T0CLKO。

输出时钟频率 = T0 溢出率/2

如果C/$\overline{T}$=0，定时器/计数器T0对内部系统时钟计数，则：

T0工作在1T模式(AUXR.7/T0x12=1)时的输出时钟频率=(SYSclk) / (256-TH0)/2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出时钟频率=(SYSclk)/12/(256-TH0)/2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (256-TH0) / 2

STC MCU Limited

;定时器0中断(下降沿中断)的测试程序，定时器/计数器0工作在计数模式中的8位自动重装模式

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T0扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：  800003751---------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------

sfr     AUXR    =       0x8e;                   //辅助寄存器
sbit    P10     =       P1^0;

//-----------------------------------------------
//外部中断服务程序
void t0int() interrupt 1                        //中断入口
{
        P10     =       !P10;                   //将测试口取反
}

void main()
{
        AUXR    =       0x80;                   //定时器0为1T模式
        TMOD    =       0x06;                   //设置定时器0为外部记数模式
        TH0  =  TL0  =  0xff;                   //设置定时器0初始值
        TR0     =       1;                      //定时器0开始工作
        ET0     =       1;                      //开定时器0中断

        EA      =       1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

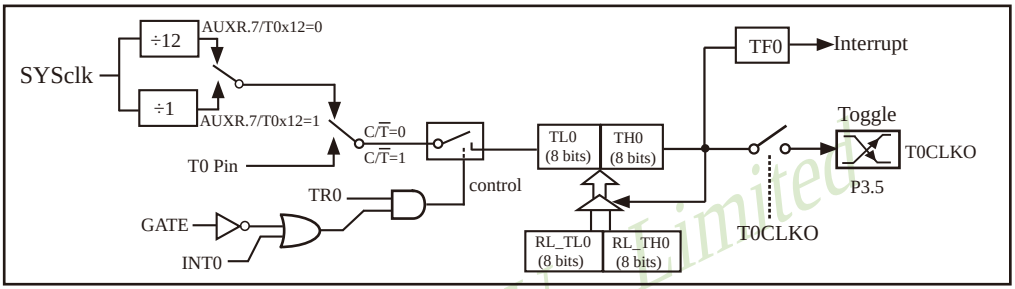
```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T0扩展为外部下降沿中断举例------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------*/


//假定测试芯片的工作频率为18.432MHz

AUXR    DATA  08EH                    //辅助寄存器
//---------------------------------------

        ORG   0000H
        LJMP  MAIN                    //复位入口

        ORG   000BH                   //中断入口
        LJMP  T0INT
//---------------------------------------

        ORG   0100H
MAIN:
        MOV   SP,     #3FH

        MOV   AUXR,   #80H            //定时器0为1T模式
        MOV   TMOD,   #06H            //设置定时器0为外部记数模式
        MOV   A,      #0FFH           //设置定时器0初始值
        MOV   TL0,    A
        MOV   TH0,    A
        SETB  TR0                     //定时器0开始工作
        SETB  ET0                     //开定时器0中断

        SETB  EA

        SJMP  $

//---------------------------------------
//外部中断服务程序

T0INT:
        CPL   P1.0                    //将测试口取反
        RETI
;---------------------------------------

        END
```

STC MCU Limited

7.2.4 模式3(不可屏蔽中断16位自动重装载, 实时操作系统用节拍定时器)

对定时器/计数器1，在模式3时，定时器1停止计数，效果与将TR1设置为0相同。

对定时器/计数器0，其工作模式3与工作模式0是一样的(下图是定时器模式3的原理图，与模式0是一样的)。唯一不同的是：当定时器/计数器0工作在模式3时，只需允许ET0/IE.1(定时器/计数器0中断允许位)不需要允许EA/IE.7(总中断使能位)就能打开定时器/计数器0的中断，此模式下的定时器/计数器0中断与总中断使能位EA无关；一旦工作在模式3下的定时器/计数器0中断被打开(ET0=1)，那么该中断是不可屏蔽的，该中断的优先级是最高的，即该中断不能被任何中断所打断，而且该中断打开后既不受EA/IE.7控制也不再受ET0控制，当EA=0或ET0=0时都不能屏蔽此中断。故将此模式称为不可屏蔽中断的16位自动重装载模式。



定时器/计数器0的模式3: 不可屏蔽中断的16位自动重装载模式

定时器/计数器0工作在模式3(不可屏蔽中断的16位自动重装载模式)，实时操作系统用系统节拍定时器。

STC创新设计，请不要抄袭，再抄袭就很无耻了

那么当定时器/计数器0工作在模式3时，如何打开定时器/计数器0的中断呢？

下面的语句可以令定时器/计数器0工作在模式3(不可屏蔽中断的16位自动重装在模式)并打开定时器/计数器0的中断(此时该中断是最高优先级，任何中断都不能屏蔽它)。

C语言设置：

```
TMOD    =    0x11;     //设置定时器为模式3(不可屏蔽中断的16位自动重装载)
TR0     =    1;        //定时器0开始计时
//EA    =    1;        //定时器0工作在模式3(不可屏蔽中断的16位自动重装载
                       //模式)时,不需要使能总中断允许位EA
ET0     =    1;        //使能定时器0工作在模式3(不可屏蔽中断的16位自动重装
                       //载模式)时的中断
```

汇编语言设置：

```
MOV     TMOD,  #00H    //设置定时器为模式0(16位自动重装载)
SETB    TR0            //定时器0开始计时
//SETB  EA             //定时器0工作在模式3(不可屏蔽中断的16位自动重装载
                       //模式)时,不需要使能总中断允许位EA
SETB    ET0            //使能定时器0工作在模式3(不可屏蔽中断的16位自动重装
                       //载模式)时的中断
```

注意：当定时器/计数器0工作在模式3(不可屏蔽中断的16位自动重装载模式)时，不需要允许EA/IE.7(总中断使能位)只需允许ET0/IE.1(定时器/计数器0中断允许位)就能打开定时器/计数器0的中断，此模式下的定时器/计数器0中断与总中断使能位EA无关。一旦此模式下的定时器/计数器0中断被打开后，该定时器/计数器0中断优先级就是最高的，它不能被其它任何中断所打断(不管是比定时器/计数器0中断优先级低的中断还是比其优先级高的中断，都不能打断此时的定时器/计数器0中断)，而且该中断打开后既不受EA/IE.7控制也不再受ET0控制了，清零EA或ET0都不能关闭此中断。

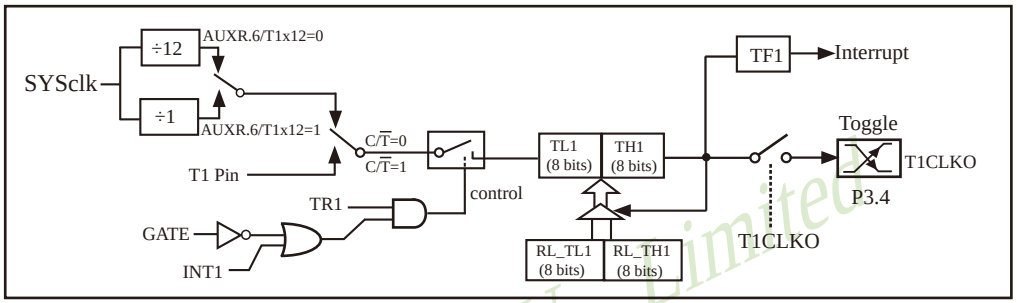
7.3 定时器/计数器1工作模式

通过对寄存器TMOD中的M1(TMOD.5)、M0(TMOD.4)的设置，定时器/计数器1有3种不同的工作模式。

7.3.1 模式0(16位自动重装载模式)及测试程序，建议只学习此模式足矣

——STC创新设计，请不要抄袭

此模式下定时器/计数器1作为可自动重装载的16位计数器，如下图所示。



定时器/计数器1的模式 0: 16位自动重装

STC创新设计，请不要抄袭，再抄袭就很无耻了

当GATE=0(TMOD.7)时，如TR1=1，则定时器计数。GATE=1时，允许由外部输入INT1控制定时器1，这样可实现脉宽测量。TR1为TCON寄存器内的控制位，TCON寄存器各位的具体功能描述见上节TCON寄存器的介绍。

当C/$\overline{T}$=0时，多路开关连接到系统时钟的分频输出，T1对内部系统时钟计数，T1工作在定时方式。当C/$\overline{T}$=1时，多路开关连接到外部脉冲输入P3.5/T1，即T1工作在计数方式。

STC15系列单片机的定时器有两种计数速率：一种是12T模式，每12个时钟加1，与传统8051单片机相同；另外一种是1T模式，每个时钟加1，速度是传统8051单片机的12倍。T1的速率由特殊功能寄存器AUXR中的T1x12决定，如果T1x12=0，T1则工作在12T模式；如果T1x12=1，T1则工作在1T模式。

定时器1有2个隐藏的寄存器RL_TH1和RL_TL1。RL_TH1与TH1共有同一个地址，RL_TL1与TL1共有同一个地址。当TR1=0即定时器/计数器1被禁止工作时，对TL1写入的内容会同时写入RL_TL1，对TH1写入的内容也会同时写入RL_TH1。当TR1=1即定时器/计数器1被允许工作时，对TL1写入内容,实际上不是写入当前寄存器TL1中,而是写入隐藏的寄存器RL_TL1中；对TH1写入内容，实际上也不是写入当前寄存器TH1中，而是写入隐藏的寄存器RL_TH1中。这样可以巧妙地实现16位重装载定时器。当读TH1和TL1的内容时，所读的内容就是TH1和TL1的内容，而不是RL_TH0和RL_TL1的内容。

当定时器1工作在模式0(TMOD[5:4]/[M1,M0]=00B)时，[TL1, TH1]的溢出不仅置位TF1，而且会自动将[RL_TL1,RL_TH1]的内容重新装入[TL1,TH1]。

当T1CLKO/INT_CLKO.1=1时，P3.4/T0管脚配置为定时器1的时钟输出T1CLKO。

输出时钟频率 = T1 溢出率/2

如果C/$\overline{T}$=0，定时器/计数器T1对内部系统时钟计数，则

T1工作在1T模式(AUXR.6/T1x12=1)时的输出**时钟频率** = (SYSclk) / (65536-[RL_TH1, RL_TL1])/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出**时钟频率**=(SYSclk)/12/(65536-[RL_TH1, RL_TL1])/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出**时钟频率** = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

7.3.1.1 定时器1的16位自动重装载模式的测试程序(C和汇编)

1. C程序：

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ---------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机定时器1的16位自动重装载模式 ------------------------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， --------------------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------------*/


//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

//-----------------------------------------

#define   FOSC    18432000L

#define   T1MS    (65536-FOSC/1000)             //1T模式，18.432MHz
//#define T1MS    (65536-FOSC/12/1000)          //12T模式，18.432MHz

sfr       AUXR    =       0x8e;                 //Auxiliary register
sbit      P10     =       P1^0;

//-----------------------------------------
```

```
/* Timer1 interrupt routine */
void tm1_isr() interrupt 3 using 1
{
        P10     =       ! P10;                  //将测试口取反
}

//-----------------------------------------

/* main program */
void main()
{
        AUXR    |=      0x40;                   //定时器1为1T模式
//      AUXR    &=      0xdf;                   //定时器1为12T模式

        TMOD    =       0x00;                   //设置定时器为模式0(16位自动重装载)
        TL1     =       T1MS;                   //初始化计时值
        TH1     =       T1MS    >> 8;
        TR1     =       1;                      //定时器1开始计时
        ET1     =       1;                      //使能定时器1中断
        EA      =        1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ------------------------------------------------------*/
/* --- 演示STC 15 系列单片机定时器1的16位自动重装载模式 -------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ---------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR    DATA    08EH            //辅助特殊功能寄存器

;-----------------------------------------
T1MS    EQU     0B800H                  //1T模式的1ms定时值(65536-18432000/1000)
//T1MS  EQU     0FA00H                  //12T模式的1ms定时值(65536-18432000/1000/12)
```

```
;-------------------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     001BH                   //中断入口
        LJMP    T1INT

;-------------------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #40H            //定时器1为1T模式
//      ANL     AUXR,   #0DFH           //定时器1为12T模式

        MOV     TMOD,   #00H            //设置定时器为模式0(16位自动重装载)

        MOV     TL1,    #LOW  T1MS      //初始化计时值
        MOV     TH1,    #HIGH T1MS
        SETB    TR1
        SETB    ET1                     //使能定时器1中断

        SETB    EA

        SJMP    $                       //程序终止

//----------------------------------------
//中断服务程序

T1INT:
        CPL     P1.0                    //将测试口取反
        RETI

;----------------------------------------

        END
```

STC MCU Limited

7.3.1.2 定时器1对系统时钟或外部引脚T1的时钟输入进行可编程分频输出的测试程序——定时器1工作在16位自动重装载模式

下面是定时器1工作在16位重装模式时对内部系统时钟或外部引脚T1/P3.5的时钟输入进行可编程时钟分频输出的程序举例(C和汇编)：

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机定时器1的可编程时钟分频输出----------------------*/
/* --- 研发顾问QQ：800003751-------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 --------------------------------------------------------------*/
/* --- Web: www.STCMCU.com ---------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ---------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef   unsigned char   BYTE;
typedef   unsigned int    WORD;

#define   FOSC    18432000L

//-----------------------------------------------
sfr       AUXR          =       0x8e;                   //辅助特殊功能寄存器
sfr       INT_CLKO      =       0x8f;                   //唤醒和时钟输出功能寄存器

sbit      T1CLKO        =       P3^4;                   //定时器1的时钟输出脚

#define   F38_4KHz      (65536-FOSC/2/38400)            //1T模式
//#define F38_4KHz      (65536-FOSC/2/12/38400)         //12T模式

//-----------------------------------------------

void main()
{
          AUXR    |=    0x40;                           //定时器1为1T模式
//        AUXR    &=    ~0x40;                          //定时器1为12T模式

          TMOD    =     0x00;                           //设置定时器为模式1(16位自动重装载)
```

STC MCU Limited

```
        TMOD    &=      ~0x40;                  //C/T1=0, 对内部时钟进行时钟输出
//      TMOD    |=      0x40;                   //C/T1=1, 对T1引脚的外部时钟进行时钟输出

        TL1     =       F38_4KHz;               //初始化计时值
        TH1     =       F38_4KHz >> 8;
        TR1     =       1;
        INT_CLKO        =       0x02;           //使能定时器1的时钟输出功能

        while (1);                              //程序终止
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机定时器1的可编程时钟分频输出----------------------*/
/* --- 研发顾问QQ：800003751-------------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------------*/
/* --- Tel: 86-755-82948412 --------------------------------------------------------*/
/* --- Web: www.STCMCU.com ---------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/


//假定测试芯片的工作频率为18.432MHz

AUXR            DATA    08EH            //辅助特殊功能寄存器
INT_CLKO        DATA    08FH            //唤醒和时钟输出功能寄存器

T1CLKO          BIT     P3.4            //定时器1的时钟输出脚

F38_4KHz        EQU     0FF10H          //38.4KHz(1T模式下, 65536-18432000/2/38400)
//F38_4KHz      EQU     0FFECH          //38.4KHz(12T模式下, (65536-18432000/2/12/38400)

        ORG     0000H
        LJMP    MAIN                    //复位入口

//-----------------------------------------------
```

```
        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #40H                    //定时器1为1T模式
//      ANL     AUXR,   #0BFH                   //定时器1为12T模式

        MOV     TMOD,   #00H                    //设置定时器为模式0(16位自动重装载)

        ANL     TMOD,   #0BFH                   //C/T1=0, 对内部时钟进行时钟输出
//      ORL     TMOD,   #40H                    //C/T1=1, 对T1引脚的外部时钟进行时钟输出

        MOV     TL1,    #LOW F38_4KHz           //初始化计时值
        MOV     TH1,    #HIGH F38_4KHz
        SETB    TR1
        MOV     INT_CLKO,       #02H            //使能定时器1的时钟输出功能

        SJMP    $                               //程序终止

;-------------------------------------------------

        END
```

STC MCU Limited

7.3.1.3 定时器1模式0(16位自动重载模式)作串口1波特率发生器的测试程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器1用作串口1的波特率发生器举例---------------------- */
/* --- 研发顾问QQ：800003751----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----*/
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define  FOSC    18432000L                  //系统频率
#define  BAUD    115200                     //串口波特率

#define  NONE_PARITY     0                  //无校验
#define  ODD_PARITY      1                  //奇校验
#define  EVEN_PARITY     2                  //偶校验
#define  MARK_PARITY     3                  //标记校验
#define  SPACE_PARITY    4                  //空白校验

#define  PARITYBIT EVEN_PARITY              //定义校验位

sfr      AUXR    =       0x8e;              //辅助寄存器

sbit     P22     =       P2^2;

bit      busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
        SCON    =       0x50;               //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        SCON    =       0xda;               //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        SCON    =       0xd2;               //9位可变波特率,校验位初始为0
#endif
```

STC MCU Limited

```
	AUXR	=	0x40;				//定时器1为1T模式
	TMOD	=	0x00;				//定时器1为模式0(16位自动重载)
	TL1	=	(65536 - (FOSC/32/BAUD));	//设置波特率重装值
	TH1	=	(65536 - (FOSC/32/BAUD))>>8;
	TR1	=	1;				//定时器1开始启动
	ES	=	1;				//使能串口中断
	EA	=	1;

	SendString("STC15F2K60S2\r\nUart Test !\r\n");
	while(1);
}

/*----------------------------
UART 中断服务程序
----------------------------*/
void Uart() interrupt 4 using 1
{
	if (RI)
	{
		RI = 0;				//清除RI位
		P0 = SBUF;			//P0显示串口数据
		P22 = RB8;			//P2.2显示校验位
	}
	if (TI)
	{
		TI = 0;				//清除TI位
		busy = 0;			//清忙标志
	}
}

/*----------------------------
发送串口数据
----------------------------*/
void SendData(BYTE dat)
{
	while (busy);				//等待前面的数据发送完成
	ACC = dat;				//获取校验位P (PSW.0)
	if (P)					//根据P来设置校验位
	{
	#if (PARITYBIT == ODD_PARITY)
		TB8 = 0;			//设置校验位为0
	#elif (PARITYBIT == EVEN_PARITY)
		TB8 = 1;			//设置校验位为1
	#endif
	}
	else
	{
	#if (PARITYBIT == ODD_PARITY)
		TB8 = 1;			//设置校验位为1
```

STC MCU Limited

```
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 0;                            //设置校验位为0
        #endif
        }
        busy = 1;
        SBUF = ACC;                             //写数据到UART数据寄存器
}

/*---------------------------
发送字符串
---------------------------*/
void SendString(char *s)
{
        while (*s)                              //检测字符串结束标志
        {
            SendData(*s++);                     //发送当前字符
        }
}
```

2. 汇编程序：

```
/*------------------------------------------------------------------------------------------------------ */
/* --- STC MCU  Limited. ------------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器1用作串口1的波特率发生器举例---------------------------------------------- */
/* --- 研发顾问QQ：  800003751-------------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ----------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                      */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                      */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*------------------------------------------------------------------------------------------------------ */

//假定测试芯片的工作频率为18.432MHz

#define   NONE_PARITY   0                   //无校验
#define   ODD_PARITY    1                   //奇校验
#define   EVEN_PARITY   2                   //偶校验
#define   MARK_PARITY   3                   //标记校验
#define   SPACE_PARITY  4                   //空白校验

#define   PARITYBIT EVEN_PARITY             //定义校验位

//---------------------------------------

AUXR    EQU     08EH                        //辅助寄存器

BUSY    BIT     20H.0                       //忙标志位

//---------------------------------------
```

STC MCU Limited

```
        ORG     0000H
        LJMP    MAIN

        ORG     0023H
        LJMP    UART_ISR
//----------------------------------------
        ORG     0100H
MAIN:
        CLR     BUSY
        CLR     EA
        MOV     SP,     #3FH

#if (PARITYBIT == NONE_PARITY)
        MOV     SCON,   #50H            //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        MOV     SCON,   #0DAH           //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     SCON,   #0D2H           //9位可变波特率,校验位初始为0
#endif

//---------------------------------
        MOV     AUXR,   #40H            //定时器1为1T模式
        MOV     TMOD,   #00H            //定时器1为模式0(16位自动重载)
        MOV     TL1,    #0FBH           //设置波特率重装值(65536-18432000/32/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                     //定时器1开始运行
        SETB    ES                      //使能串口中断
        SETB    EA

        MOV     DPTR,   #TESTSTR        //发送测试字符串
        LCALL   SENDSTRING

        SJMP    $
;----------------------------------------
TESTSTR:
        DB  "STC15F2K60S2 Uart1 Test !",0DH,0AH,0

;/*---------------------------
;UART 中断服务程序
;---------------------------*/
UART_ISR:
        PUSH    ACC
        PUSH    PSW
        JNB     RI,     CHECKTI                 //检测RI位
        CLR     RI                              //清除RI位
        MOV     P0,     SBUF                    //P0显示串口数据
        MOV     C,      RB8
        MOV     P2.2,   C                       //P2.2显示校验位
```

STC MCU Limited

```
CHECKTI:
        JNB     TI,     ISR_EXIT            //检测TI位
        CLR     TI                          //清除TI位
        CLR     BUSY                        //清忙标志
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

;/*----------------------------
;发送串口数据
;----------------------------*/
SENDDATA:
        JB      BUSY,   $                   //等待前面的数据发送完成
        MOV     ACC,    A                   //获取校验位P (PSW.0)
        JNB     P,      EVEN1INACC          //根据P来设置校验位
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        CLR     TB8                         //设置校验位为0
#elif (PARITYBIT == EVEN_PARITY)
        SETB    TB8                         //设置校验位为1
#endif
        SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        SETB    TB8                         //设置校验位为1
#elif (PARITYBIT == EVEN_PARITY)
        CLR     TB8                         //设置校验位为0
#endif
PARITYBITOK:                                //校验位设置完成
        SETB    BUSY
        MOV     SBUF,   A                   //写数据到UART数据寄存器
        RET

;/*----------------------------
;发送字符串
//----------------------------*/
SENDSTRING:
        CLR     A
        MOVC    A,      @A+DPTR             //读取字符
        JZ      STRINGEND                   //检测字符串结束标志
        INC     DPTR                        //字符串地址+1
        LCALL   SENDDATA                    //发送当前字符
        SJMP    SENDSTRING
STRINGEND:
        RET
//-----------------------------------------
        END
```

STC MCU Limited

7.3.1.4 T1的16位自动重装载模式扩展为外部下降沿中断的测试程序(C和汇编)
——利用T1的外部计数方式

;定时器1中断(下降沿中断)的测试程序，定时器/计数器1工作在计数模式中的16位自动重装载模式

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T1扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------

sfr AUXR = 0x8e;                        //辅助寄存器
sbit P10 = P1^0;

//-----------------------------------------
//外部中断服务程序
void   t1int()  interrupt 3                     //中断入口
{
        P10     =       !P10;           //将测试口取反
}

void   main()
{
        AUXR    =       0x40;   //定时器1为1T模式
        TMOD    =       0x40;   //设置定时器1为外部记数模式，工作在16位自动重装载模式
        TH1     =  TL1 =  0xff;   //设置定时器1初始值
        TR1     =       1;      //定时器1开始工作
        ET1     =       1;       //开定时器1中断

        EA      =       1;

        while (1);
}
```

STC MCU Limited

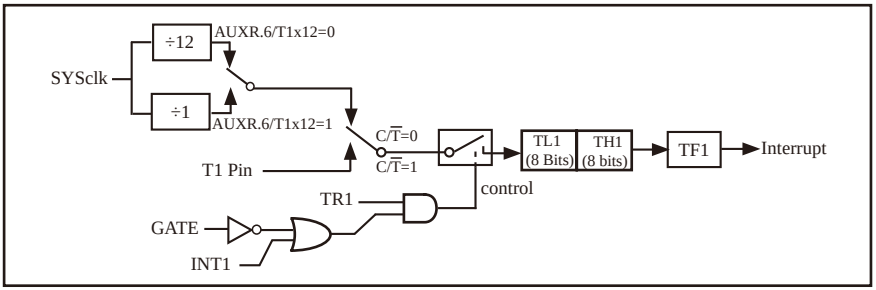
2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T1扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR    DATA    08EH                    //辅助寄存器
//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     001BH                   //中断入口
        LJMP    T1INT
//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        MOV     AUXR,   #40H            //定时器1为1T模式
        MOV     TMOD,   #40H            //设置定时器1为外部记数模式，工作在16位重装载模式
        MOV     A,      #0FFH           //设置定时器1初始值
        MOV     TL1,    A
        MOV     TH1,    A
        SETB    TR1                     //定时器1开始工作
        SETB    ET1                     //开定时器1中断

        SETB    EA

        SJMP    $

//-----------------------------------------
//外部中断服务程序

T1INT:
        CPL     P1.0                    //将测试口取反
        RETI
;-----------------------------------------

        END
```

STC MCU Limited

7.3.2 模式1(16位不可重装载模式)，不建议学习

此模式下定时器/计数器1作为16位定时器，如下图所示。



定时器/计数器1的模式 1: 16位不可重装载

此模式下，定时器1配置为16位不可重装在模式，由TL1的8位和TH1的8位所构成。TL1的8位溢出向TH1进位，TH1计数溢出置位TCON中的溢出标志位TF1。

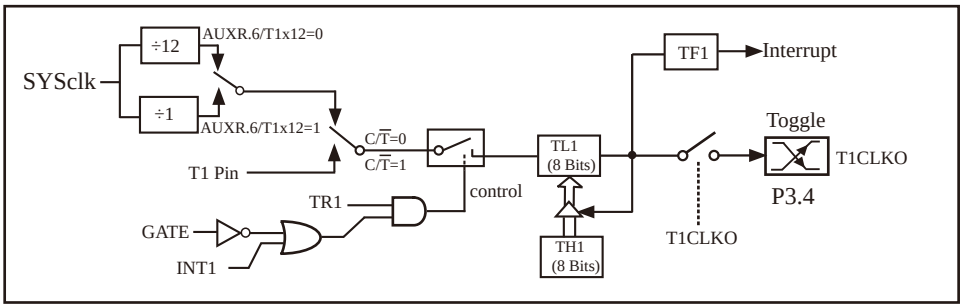
当GATE=0(TMOD.7)时，如TR1=1，则定时器计数。GATE=1时，允许由外部输入INT1控制定时器1，这样可实现脉宽测量。TR1为TCON寄存器内的控制位，TCON寄存器各位的具体功能描述见上节TCON寄存器的介绍。

当C/$\overline{T}$=0时，多路开关连接到系统时钟的分频输出，T1对内部系统时钟计数，T1工作在定时方式。当C/$\overline{T}$=1时，多路开关连接到外部脉冲输入P3.5/T1，即T1工作在计数方式。

STC15系列单片机的定时器有两种计数速率：一种是12T模式，每12个时钟加1，与传统8051单片机相同；另外一种是1T模式，每个时钟加1，速度是传统8051单片机的12倍。T1的速率由特殊功能寄存器AUXR中的T1x12决定，如果T1x12=0，T1则工作在12T模式；如果T1x12=1，T1则工作在1T模式。

7.3.3 模式2(8位自动重装载模式)，不建议学习

此模式下定时器/计数器1作为可自动重装载的8位计数器，如下图所示。



定时器/计数器1的模式 2: 8位自动重装载

TL1的溢出不仅置位TF1，而且将TH1内容重新装入TL1，TH1内容由软件预置，重装时TH1内容不变。

当T1CLKO/INT_CLKO.1=1时，P3.4/T0管脚配置为定时器1的时钟输出T1CLKO。

输出时钟频率 = T1 溢出率/2

如果C/$\overline{T}$=0，定时器/计数器T1对内部系统时钟计数，则

T1工作在1T模式(AUXR.6/T1x12=1)时的输出时钟频率=(SYSclk) / (256-TH1)/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出时钟频率=(SYSclk)/12/(256-TH1)/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1) / 2

STC MCU Limited

7.3.3.1 定时器1模式2(8位自动重载模式)作串口1波特率发生器的测试程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器1用作串口1的波特率发生器举例---------------------- */
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序---------------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序---------------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz#include "reg51.h"

#include "intrins.h"

typedef unsigned char        BYTE;
typedef unsigned int         WORD;

#define   FOSC    18432000L                 //系统频率
#define   BAUD    115200                    //串口波特率

#define   NONE_PARITY       0       //无校验
#define   ODD_PARITY        1       //奇校验
#define   EVEN_PARITY       2       //偶校验
#define   MARK_PARITY       3       //标记校验
#define   SPACE_PARITY      4       //空白校验

#define PARITYBIT EVEN_PARITY   //定义校验位

sfr       AUXR    =         0x8e;             //辅助寄存器

sbit      P22     =         P2^2;

bit       busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
        SCON = 0x50;                        //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        SCON = 0xda;                        //9位可变波特率,校验位初始为1
```

STC MCU Limited

```
#elif (PARITYBIT == SPACE_PARITY)
	SCON	=	0xd2;				//9位可变波特率,校验位初始为0
#endif

	AUXR	=	0x40;				//定时器1为1T模式
	TMOD	=	0x20;				//定时器1为模式2(8位自动重载)
	TL1	=	(256 - (FOSC/32/BAUD));		//设置波特率重装值
	TH1	=	(256 - (FOSC/32/BAUD));
	TR1	=	1;				//定时器1开始工作
	ES	=	1;				//使能串口中断
	EA	=	1;

	SendString("STC15F2K60S2\r\nUart Test !\r\n");
	while(1);
}

/*----------------------------
UART 中断服务程序
-----------------------------*/
void Uart() interrupt 4 using 1
{
	if (RI)
	{
		RI	=	0;			//清除RI位
		P0	=	SBUF;			//P0显示串口数据
		P22	=	RB8;			//P2.2显示校验位
	}
	 if (TI)
	{
		TI	=	0;			//清除TI位
		busy	=	0;			//清忙标志
	}
}

/*----------------------------
发送串口数据
----------------------------*/
void SendData(BYTE dat)
{
	while (busy);					//等待前面的数据发送完成
	ACC = dat;					//获取校验位P (PSW.0)
	if (P)						//根据P来设置校验位
	{
	#if (PARITYBIT == ODD_PARITY)
		TB8 = 0;				//设置校验位为0
	#elif (PARITYBIT == EVEN_PARITY)
		TB8 = 1;				//设置校验位为1
	#endif
	}
```

STC MCU Limited

```
        else
        {
        #if (PARITYBIT == ODD_PARITY)
                TB8 = 1;                                //设置校验位为1
        #elif (PARITYBIT == EVEN_PARITY)
                TB8 = 0;                                //设置校验位为0
        #endif
        }
        busy = 1;
        SBUF = ACC;                                     //写数据到UART数据寄存器
}

/*----------------------------
发送字符串
----------------------------*/
void SendString(char *s)
{
        while (*s)                                      //检测字符串结束标志
        {
                SendData(*s++);                         //发送当前字符
        }
}
```

2. 汇编程序：

```
/*------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器1用作串口1的波特率发生器举例---------------------------------------- */
/* --- 研发顾问QQ：800003751----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ----------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序---------------------------------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序---------------------------------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-----------*/
/*------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#define   NONE_PARITY   0                 //无校验
#define   ODD_PARITY    1                 //奇校验
#define   EVEN_PARITY   2                 //偶校验
#define   MARK_PARITY   3                 //标记校验
#define   SPACE_PARITY  4                 //空白校验

#define   PARITYBIT  EVEN_PARITY          //定义校验位

//---------------------------------------
```

STC MCU Limited

```
AUXR    EQU     08EH                    //辅助寄存器
BUSY    BIT     20H.0                   //忙标志位

//----------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     0023H
        LJMP    UART_ISR
//----------------------------------------
        ORG     0100H
MAIN:
        CLR     BUSY
        CLR     EA
        MOV     SP,     #3FH

#if (PARITYBIT == NONE_PARITY)
        MOV     SCON,   #50H            //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        MOV     SCON,   #0DAH           //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     SCON,   #0D2H           //9位可变波特率,校验位初始为0
#endif

//---------------------------------
        MOV     AUXR,   #40H            //定时器1为1T模式
        MOV     TMOD,   #20H            //定时器1为模式2(8位自动重载)
        MOV     TL1,    #0FBH           //设置波特率重装值(256-18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1                     //定时器1开始运行
        SETB    ES                      //使能串口中断
        SETB    EA

        MOV     DPTR,   #TESTSTR        //发送测试字符串
        LCALL   SENDSTRING

        SJMP    $
;------------------------------------------
TESTSTR:
        DB  "STC15F2K60S2 Uart1 Test !",0DH,0AH,0

;/*--------------------------
;UART 中断服务程序
;--------------------------*/
UART_ISR:
        PUSH    ACC
        PUSH    PSW
        JNB     RI,     CHECKTI         //检测RI位
        CLR     RI                      //清除RI位
        MOV     P0,     SBUF            //P0显示串口数据
        MOV     C,      RB8
```

STC MCU Limited

```
        MOV     P2.2,   C                       //P2.2显示校验位
CHECKTI:
        JNB     TI,     ISR_EXIT                //检测TI位
        CLR     TI                              //清除TI位
        CLR     BUSY                            //清忙标志
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

;/*---------------------------
;发送串口数据
;---------------------------*/
SENDDATA:
        JB      BUSY,   $                       //等待前面的数据发送完成
        MOV     ACC,    A                       //获取校验位P (PSW.0)
        JNB     P,      EVEN1INACC              //根据P来设置校验位
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        CLR     TB8                             //设置校验位为0
#elif (PARITYBIT == EVEN_PARITY)
        SETB    TB8                             //设置校验位为1
#endif
        SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        SETB    TB8                             //设置校验位为1
#elif (PARITYBIT == EVEN_PARITY)
        CLR     TB8                             //设置校验位为0
#endif
PARITYBITOK:                                    //校验位设置完成
        SETB    BUSY
        MOV     SBUF,   A                       //写数据到UART数据寄存器
        RET

;/*---------------------------
;发送字符串
//---------------------------*/
SENDSTRING:
        CLR     A
        MOVC    A,      @A+DPTR                 //读取字符
        JZ      STRINGEND                       //检测字符串结束标志
        INC     DPTR                            //字符串地址+1
        LCALL   SENDDATA                        //发送当前字符
        SJMP    SENDSTRING
STRINGEND:
        RET
//-----------------------------------------
        END
```

STC MCU Limited

7.3.3.2 T1的8位自动重装载模式扩展为外部下降沿中断的测试程序(C和汇编)

;定时器1中断(下降沿中断)的测试程序，定时器/计数器1工作在计数模式中的8位自动重装载模式

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T1扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------

sfr     AUXR    =       0x8e;                   //辅助寄存器
sbit    P10     =       P1^0;

//-----------------------------------------
//外部中断服务程序
void t1int() interrupt 3                        //中断入口
{
        P10     =       !P10;                   //将测试口取反
}

void main()
{
        AUXR    =       0x40;           //定时器1为1T模式
        TMOD    =       0x60;           //设置定时器1为外部记数模式，工作在8位自动重装载模式
        TH1 =   TL1 =   0xff;           //设置定时器1初始值
        TR1     =       1;              //定时器1开始工作
        ET1     =       1;              //开定时器1中断

        EA      =       1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T1扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR    DATA    08EH                    //辅助寄存器
//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     001BH                   //中断入口
        LJMP    T1INT
//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        MOV     AUXR,   #40H            //定时器1为1T模式
        MOV     TMOD,   #60H            //设置定时器1为外部记数模式
        MOV     A,      #0FFH           //设置定时器1初始值
        MOV     TL1,    A
        MOV     TH1,    A
        SETB    TR1                     //定时器1开始工作
        SETB    ET1                     //开定时器1中断

        SETB    EA

        SJMP    $

//-----------------------------------------
//外部中断服务程序

T1INT:
        CPL     P1.0                    //将测试口取反
        RETI
;-----------------------------------------

        END
```

STC MCU Limited

7.4 古老的Intel 8051单片机定时器0/1应用举例

【例1】 定时/计数器应用编程，设某应用系统，选择定时/计数器1定时模式，定时时间Tc = 10ms，主频频率为12MHz，每10ms向主机请求处理。选定工作方式1。计算得计数初值：低8位初值为F0H，高8位初值为D8H。

（1）初始化程序

所谓初始化，一般在主程序中根据应用要求对定时/计数器进行功能选择及参数设定等预置程序，本例初始化程序如下：

```
START:
        ⋮                           ; 主程序段
        MOV     SP,     #60H        ; 设置堆栈区域
        MOV     TMOD,   #10H        ; 选择T1、定时模式，工作方式1
        MOV     TH1,    #0D8H       ; 设置高字节计数初值
        MOV     TL1,    #0F0H       ; 设置低字节计数初值
        SETB    EA                  ;
        SETB    ET1                 ; } 开中断
        ⋮                           ; 其他初始化程序
        SETB    TR1                 ; 启动T1开始计时
        ⋮                           ; 继续主程序
```

（2）中断服务程序

```
INTT1:  PUSH    A                   ;
        PUSH    DPL                 ; } 现场保护
        PUSH    DPH                 ;
        ⋮
        MOV     TL1,    #0F0H       ;
        MOV     TH1,    #0D8H       ; } 重新置初值
        ⋮                           ; 中断处理主体程序
        POP     DPH                 ;
        POP     DPL                 ; } 现场恢复
        POP     A                   ;
        RETI                        ; 返回
```

STC MCU Limited

这里展示了中断服务子程序的基本格式。STC15系列单片机的中断属于矢量中断，每一个矢量中断源只留有8个字节单元，一般是不够用的，常需用转移指令转到真正的中断服务子程序区去执行。

【例2】 利用定时/计数器0或定时/计数器1的Tx端口改造成外部中断源输入端口的应用设计。

在某些应用系统中常会出现原有的两个外部中断源INT0和INT1不够用，而定时/计数器有多余，则可将Tx用于增加的外部中断源。现选择定时/计数器1为对外部事件计数模式工作方式2（自动再装入），设置计数初值为FFH，则T1端口输入一个负跳变脉冲，计数器即回0溢出，置位对应的中断请求标志位TF1为1，向主机请求中断处理，从而达到了增加一个外部中断源的目的。应用定时/计数器1（T1）的中断矢量转入中断服务程序处理。其程序示例如下：

（1）主程序段：

```
        ORG     0000H
        AJMP    MAIN                ；转主程序
        ORG     001BH
        LJMP    INTER               ；转T1中断服务程序
        ⋮
        ORG     0100                ；主程序入口
MAIN:   …
        ⋮
        MOV     SP,     #60H        ；设置堆栈区
        MOV     TMOD,   #60H        ；设置定时/计数器1，计数方式2
        MOV     TL1,    #0FFH       ；设置计数常数
        MOV     TH1,    #0FFH
        SETB    EA                  ；开中断
        SETB    ET1                 ；开定时/计数器1中断
        SETB    TR1                 ；启动定时/计数器1计数
        ⋮
```

STC MCU Limited

（2）中断服务程序（具体处理程序略）

```
                ORG     1000H
INTER:          PUSH    A                ;
                PUSH    DPL              ;  } 现场入栈保护
                PUSH    DPH              ;
                ⋮
                ⋮                        ;  } 中断处理主体程序
                ⋮                        ;
                POP     DPH              ;
                POP     DPL              ;  } 现场出栈复原
                POP     A                ;
                RETI                     ; 返回
```

这是中断服务程序的基本格式。

【例5】 某应用系统需通过P1.0和P1.1分别输出周期为200μs和400μs的方波。为此，系统选用定时器/计数器0（T0），定时方式3，主频为6MHz，TP=2μs，经计算得定时常数为9CH和38H。

本例程序段编制如下：

（1）初始化程序段

```
                ⋮
PLT0:   MOV     TMOD,#03H        ; 设置T0定时方式3
        MOV     TL0，#9CH        ; 设置TL0初值
        MOV     TH0，#38H        ; 设置TH0初值
        SETB    EA               ;
        SETB    ET0              ;  } 开中断
        SETB    ET1              ;
        SETB    TR0              ; 启动
        SETB    TR1              ; 启动
        ⋮
```

STC MCU Limited

(2)中断服务程序段

1)

```
INT0P:  ⋮
        ⋮
        MOV     TL0,    #9CH            ; 重新设置初值
        CPL     P1.0                    ; 对P1.0输出信号取反
        ⋮
        RETI                            ; 返回
```

2)

```
INT1P   ⋮
        ⋮
        MOV     TH0,    #38H            ; 重新设置初值
        CPL     P1.1                    ; 对P1.1输出信号取反
        ⋮
        RETI                            ; 返回
```

STC MCU Limited

在实际应用中应注意的问题如下。

（1）定时/计数器的实时性

定时/计数器启动计数后，当计满回0溢出向主机请求中断处理，由内部硬件自动进行。但从回0溢出请求中断到主机响应中断并作出处理存在时间延迟，且这种延时随中断请求时的现场环境的不同而不同，一般需延时3个机器周期以上，这就给实时处理带来误差。大多数应用场合可忽略不计，但对某些要求实时性苛刻的场合，应采用补偿措施。

这种由中断响应引起的时间延时，对定时/计数器工作于方式0或1而言有两种含义：一是由于中断响应延时而引起的实时处理的误差；二是如需多次且连续不间断地定时/计数，由于中断响应延时，则在中断服务程序中再置计数初值时已延误了若干个计数值而引起误差，特别是用于定时就更明显。

例如选用定时方式1设置系统时钟，由于上述原因就会产生实时误差。这种场合应采用动态补偿办法以减少系统始终误差。所谓动态补偿，即在中断服务程序中对THx、TLx重新置计数初值时，应将THx、TLx从回0溢出又重新从0开始继续计数的值读出，并补偿到原计数初值中去进行重新设置。可考虑如下补偿方法：

```
    ⋮
CLR     EA              ；禁止中断
MOV     A,      TLx     ；读TLx中已计数值
ADD     A,      #LOW    ；LOW为原低字节计数初值
MOV     TLx,    A       ；设置低字节计数初值
MOV     A,      #HIGH   ；原高字节计数初值送A
ADDC    A,      THx     ；高字节计数初值补偿
MOV     THx,    A       ；置高字节计数初值
SETB    EA              ；开中断
    ⋮
```

（2）动态读取运行中的计数值

在动态读取运行中的定时/计数器的计数值时，如果不加注意，就可能出错。这是因为不可能在同一时刻同时读取THx和TLx中的计数值。比如，先读TLx后读THx，因为定时/计数器处于运行状态，在读TLx时尚未产生向THx进位，而在读THx前已产生进位，这时读得的THx就不对了；同样，先读THx后读TLx也可能出错。

一种可避免读错的方法是：先读THx，后读TLx，将两次读得的THx进行比较；若两次读得的值相等，则可确定读的值是正确的，否则重复上述过程，重复读得的值一般不会再错。此法的软件编程如下：

```
RDTM:
        MOV     A,      THx             ；读取THx存A中
        MOV     R0,     TLx             ；读取TLx存R0中
        CJNE    A,      THx，RDTM       ；比较两次THx值,若相等,则读得的
                                        ；值正确,程序往下执行,否则重读
        MOV     R1,     A               ；将THx存于R1中
          ⋮
```

STC MCU Limited

7.5 定时器/计数器2及其应用(STC创新设计，请不要抄袭)

T2的工作模式固定为16位自动重装载模式，T2可以当定时器/计数器用，也可以当可编程时钟输出和串口的波特率发生器。

下面首先介绍与定时器/计数器2相关的寄存器：

7.5.1 定时器/计数器2的相关特殊功能寄存器

与定时器/计数器2有关的特殊功能寄存器：

符号	描述	地址	位地址及其符号 MSB LSB	复位值
T2H	定时器2高8位寄存器	D6H		0000 0000B
T2L	定时器2低8位寄存器	D7H		0000 0000B
AUXR	辅助寄存器	8EH	T0x12 T1x12 UART_M0x6 T2R T2_C/T T2x12 EXTRAM S1ST2	0000 0001B
INT_CLKO AUXR2	外部中断允许和时钟输出寄存器	8FH	- EX4 EX3 EX2 - T2CLKO T1CLKO T0CLKO	x000 x000B
IE2	Interrupt Enable register	AFH	- ET4 ET3 ES4 ES3 ET2 ESPI ES2	x000 0000B

1. 定时器2的控制寄存器：辅助寄存器AUXR

STC15系列单片机是 1T 的8051单片机，为兼容传统8051，定时器0、定时器1, 和定时器2复位后是传统8051的速度，即12分频，这是为了兼容传统8051。但也可不进行12分频，通过设置新增加的特殊功能寄存器AUXR, 将T0, T1, T2设置为1T。普通111条机器指令执行速度是固定的，快3到24倍，无法改变。

AUXR格式如下：

AUXR：辅助寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T2R：定时器2允许控制位
　　0，不允许定时器2运行；
　　1，允许定时器2运行

T2_C/T：控制定时器2用作定时器或计数器。
　　0，用作定时器(对内部系统时钟进行计数)；
　　1，用作计数器(对引脚T2/P3.1的外部脉冲进行计数)

T2x12：定时器2速度控制位
　　0，定时器2是传统8051速度，12分频；
　　1，定时器2的速度是传统8051的12倍，不分频

如果串口1或串口2用T2作为波特率发生器，则由T2x12决定串口1或串口2是12T还是1T.

STC MCU Limited

T0x12：定时器0速度控制位

0，定时器0是传统8051速度，12分频；

1，定时器0的速度是传统8051的12倍，不分频

T1x12：定时器1速度控制位

0，定时器1是传统8051速度，12分频；

1，定时器1的速度是传统8051的12倍，不分频

如果UART1/串口1用T1作为波特率发生器，则由T1x12决定UART1/串口是12T还是1T

UART_M0x6：串口模式0的通信速度设置位。

0，串口1模式0的速度是传统8051单片机串口的速度，12分频；

1，串口1模式0的速度是传统8051单片机串口速度的6倍，2分频

EXTRAM：内部/外部RAM存取控制位

0，允许使用逻辑上在片外、物理上在片内的扩展RAM；

1，禁止使用逻辑上在片外、物理上在片内的扩展RAM

S1ST2：串口1(UART1)选择定时器2作波特率发生器的控制位

0，选择定时器1作为串口1(UART1)的波特率发生器；

1，选择定时器2作为串口1(UART1)的波特率发生器，此时定时器1得到释放，可以作为独立定时器使用

STC MCU Limited

2. T2的时钟输出允许控制位T2CLKO

T2CLKO/P3.0的时钟输出控制由INT_CLKO(AUXR2)寄存器中的T2CLKO位控制。T2CLKO的输出时钟频率由定时器2控制,不要允许相应的定时器中断,免得CPU反复进中断。定时器2的工作模式固定为模式0(16位自动重装载模式),在此模式下定时器2可用作时钟输出。

INT_CLKO (AUXR2)格式如下：

INT_CLKO (AUXR2)：外部中断允许和时钟输出寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T2CLKO：是否允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

1：允许将P3.0脚配置为定时器2的时钟输出T2CLKO，输出时钟频率=T2溢出率/2

如果T2_C/$\overline{T}$=0，定时器/计数器T2是对内部系统时钟计数，则：

T2工作在1T模式(AUXR.2/T2x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH2, RL_TL2])/2

T2工作在12T模式(AUXR.2/T2x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH2, RL_TL2])/2

如果T2_C/$\overline{T}$=1，定时器/计数器T2是对外部脉冲输入(P3.1/T2)计数，则：

输出时钟频率 = (T2_Pin_CLK) / (65536-[RL_TH2, RL_TL2])/2

0：不允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

T0CLKO ： 是否允许将P3.5/T1脚配置为定时器0(T0)的时钟输出T0CLKO

1，将P3.5/T1管脚配置为定时器0的时钟输出T0CLKO，输出时钟频率=T0溢出率/2

若定时器/计数器T0工作在定时器模式0(16位自动重装载模式)时，

如果C/$\overline{T}$=0，定时器/计数器T0是对内部系统时钟计数，则：

T0工作在1T模式(AUXR.7/T0x12=1)时的输出频率 = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH0, RL_TL0])/2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

若定时器/计数器T0工作在定时器模式2(8位自动重装模式)，

如果C/$\overline{T}$=0,定时器/计数器T1是对内部系统时钟计数，则：

T0工作在1T模式(AUXR.7/T0x12=1)时的输出频率 = (SYSclk) / (256-TH0) / 2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出频率 = (SYSclk) / 12 / (256-TH0) / 2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (256-TH0) / 2

0，不允许P3.5/T1管脚被配置为定时器0的时钟输出

T1CLKO：是否允许将P3.4/T0脚配置为定时器1(T1)的时钟输出T1CLKO

1，将P3.4/T0管脚配置为定时器1的时钟输出T1CLKO，输出时钟频率= T1溢出率/2

若定时器/计数器T1工作在定时器模式0(16位自动重装载模式)，

如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH1, RL_TL1])/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH1, RL_TL1])/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

若定时器/计数器T1工作在模式2(8位自动重装模式)，

如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (256-TH1)/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk)/12/(256-TH1)/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1) / 2

0，不允许P3.4/T0管脚被配置为定时器1的时钟输出

EX4： 外部中断4($\overline{INT4}$)中断允许位，EX4=1允许中断，EX4=0禁止中断。外部中断4($\overline{INT4}$)只能下降沿触发。

EX3： 外部中断3($\overline{INT3}$)中断允许位，EX3=1允许中断，EX3=0禁止中断。外部中断3($\overline{INT3}$)也只能下降沿触发。

EX2： 外部中断2($\overline{INT2}$)中断允许位，EX2=1允许中断，EX2=0禁止中断。外部中断2($\overline{INT2}$)同样只能下降沿触发。

STC MCU Limited

3. T2的中断允许控制位ET2

IE2：中断允许寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4：定时器4的中断允许位。
1，允许定时器4产生中断；
0，禁止定时器4产生中断。

ET3：定时器3的中断允许位。
1，允许定时器3产生中断；
0，禁止定时器3产生中断。

ES4：串行口4中断允许位。
1，允许串行口4中断；
0，禁止串行口4中断

ES3：串行口3中断允许位。
1，允许串行口3中断；
0，禁止串行口3中断。

ET2：定时器2的中断允许位。
1，允许定时器2产生中断；
0，禁止定时器2产生中断。

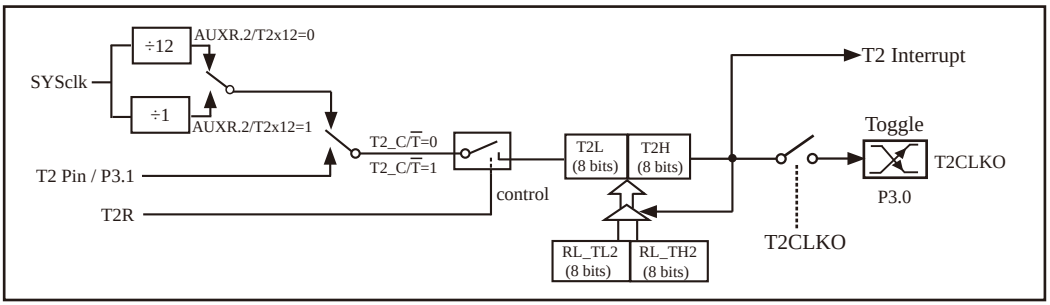
ESPI：SPI中断允许位。
1，允许SPI中断；
0，禁止SPI中断。

ES2：串行口2中断允许位。
1，允许串行口2中断；
0，禁止串行口2中断。

STC MCU Limited

7.5.2 定时器/计数器2作定时器及测试程序(C和汇编)

定时器/计数器2的原理框图如下：



定时器/计数器2的工作模式: 16位自动重装

STC创新设计，请不要抄袭，再抄袭就很无耻了

T2R/AUXR.4为AUXR寄存器内的控制位，AUXR寄存器各位的具体功能描述见上节AUXR寄存器的介绍。

当T2_C/$\overline{T}$=0时，多路开关连接到系统时钟输出，T2对内部系统时钟计数，T2工作在定时方式。当T2_C/$\overline{T}$=1时，多路开关连接到外部脉冲输入P3.1/T2，即T2工作在计数方式。

STC15系列单片机的定时器2有两种计数速率：一种是12T模式，每12个时钟加1，与传统8051单片机相同；另外一种是1T模式，每个时钟加1，速度是传统8051单片机的12倍。T2的速率由特殊功能寄存器AUXR中的T2x12决定，如果T2x12=0，T2则工作在12T模式；如果T2x12=1，T2则工作在1T模式。

定时器2有2个隐藏的寄存器RL_TH2和RL_TL2。RL_TH2与T2H共有同一个地址，RL_TL2与T2L共有同一个地址。当T2R=0即定时器/计数器2被禁止工作时，对T2L写入的内容会同时写入RL_TL2，对T2H写入的内容也会同时写入RL_TH2。当T2R=1即定时器/计数器2被允许工作时，对T2L写入内容，实际上不是写入当前寄存器T2L中,而是写入隐藏的寄存器RL_TL2中；对T2H写入内容，实际上也不是写入当前寄存器T2H中，而是写入隐藏的寄存器RL_TH2。当读T2H和T2L的内容时，所读的内容就是T2H和T2L的内容，而不是RL_TH2和RL_TL2的内容。

这样可以巧妙地实现16位重装载定时器。[T2L,T2H]的溢出不仅置位被隐藏的中断请求标志位(定时器2的中断请求标志位对用户不可见)，使CPU转去执行定时器2的中断程序，而且会自动将[RL_TL2,RL_TH2]的内容重新装入[T2L,T2H]。

7.5.2.1 定时器2的16位自动重载模式的测试程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 定时器2的16位自动重装载模式举例----------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

//-----------------------------------------

/* define constants */
#define    FOSC 18432000L

#define    T38_4KHz          (256-18432000/12/38400/2)          //38.4KHz

/* define SFR */

sfr        IE2       =       0xAF;                              //(IE2.2)timer2 interrupt control bit
sfr        AUXR      =       0x8E;
sfr        T2H       =       0xD6;
sfr        T2H       =       0xD7;

sbit       TEST_PIN          =          P0^0;                   //test pin

//-----------------------------------------

/* Timer2 interrupt routine */
void t2_isr() interrupt 12 using 1
{
           TEST_PIN =        !TEST_PIN;
}
```

STC MCU Limited

```
//---------------------------------------------
/* main program */
void main()
{
        T2L     =       T38_4KHz;                       //set timer2 reload value
        T2H     =       T38_4KH   >> 8;
        AUXR    |=      0x10;                           //timer2 start run
        IE2     |=      0x04;                           //enable timer2 interrupt
        EA      =       1;                              //open global interrupt switch

        while (1);                                      //loop
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 定时器2的16位自动重装载模式举例-------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

IE2     DATA    0AFH                    //中断使能寄存器2
AUXR    DATA    08EH                    //辅助寄存器
T2H     DATA    0D6H                    //定时器2高8位
T2L     DATA    0D7H                    //定时器2低8位

F38_4KHz        EQU     0FF10H          //38.4KHz(1T模式下, 65536-18432000/2/38400)

//-----------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     0063H                   //中断入口
        LJMP    T2INT
```

```
//-----------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #04H                    //定时器2为1T模式

        MOV     T2L,    #LOW F38_4KHz           //初始化计时值
        MOV     T2H,    #HIGH F38_4KHz

        ORL     AUXR,   #10H                    //定时器2开始计时

        ORL     IE2,    #04H                    //开定时器2中断

        SETB    EA

        SJMP    $

//-----------------------------------------
//外部中断服务程序

T2INT:
        CPL   P1.0                              //将测试口取反

//      ANL   IE2,      #0FBH                   //若需要手动清除中断标志,可先关闭中断,
                                                //此时系统会自动清除内部的中断标志
//      ORL   IE2,      #04H                    //然后再开中断即可

        RETI

;-----------------------------------------

        END
```

STC MCU Limited

7.5.2.2 定时器2扩展为外部下降沿中断的的测试程序(C和汇编)

;定时器2中断(下降沿中断)的测试程序，定时器/计数器2工作在计数模式中的16位自动重装载模式

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T2扩展为外部下降沿中断举例-----------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------
sfr     IE2     =       0xaf;           //中断使能寄存器2
sfr     AUXR    =       0x8e;           //辅助寄存器
sfr     T2H     =       0xD6;           //定时器2高8位
sfr     T2L     =       0xD7;           //定时器2低8位

sbit    P10     =       P1^0;
//-----------------------------------------
//中断服务程序
void t2int() interrupt 12                       //中断入口
{
        P10     =       !P10;                   //将测试口取反

//      IE2     &=      ~0x04;                  //若需要手动清除中断标志,可先关闭中断,
                                                //此时系统会自动清除内部的中断标志
//      IE2     |=      0x04;                   //然后再开中断即可
}

void main()
{
        AUXR    |=      0x04;           //定时器2为1T模式
        AUXR    |=      0x08;           //T2_C/T=1, T2(P3.1)引脚为时钟源
        T2H  =  T2L  =  0xff;           //初始化计时值
        AUXR    |=      0x10;           //定时器2开始计时

        IE2     |=      0x04;                   //开定时器2中断

        EA      =       1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 T2扩展为外部下降沿中断举例------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

IE2     DATA    0AFH                    //中断使能寄存器2
AUXR    DATA    08EH                    //辅助寄存器
T2H     DATA    0D6H                    //定时器2高8位
T2L     DATA    0D7H                    //定时器2低8位

//---------------------------------------

        ORG     0000H
        LJMP    MAIN                    //复位入口

        ORG     0063H                   //中断入口
        LJMP    T2INT
//---------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #04H            //定时器2为1T模式
        ORL     AUXR,   #08H            //T2_C/T=1, T2(P3.1)引脚为时钟源

        MOV     A,      #0FFH           //初始化计时值
        MOV     T2L,    A
        MOV     T2H,    A

        ORL     AUXR,   #10H            //定时器2开始计时

        ORL     IE2,    #04H            //开定时器2中断

        SETB    EA

        SJMP    $
//---------------------------------------
```

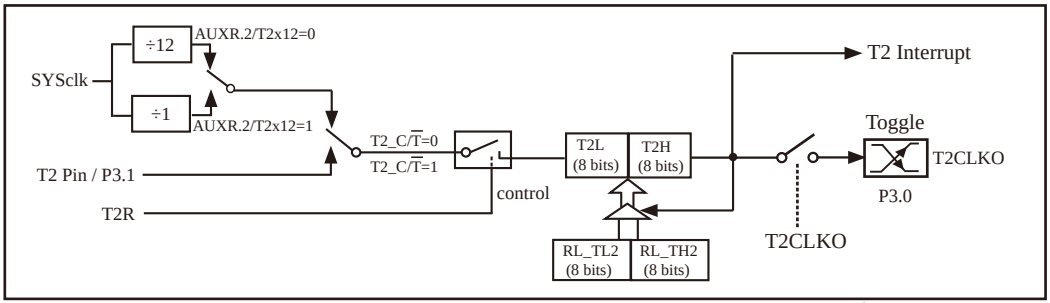
STC MCU Limited

```
//外部中断服务程序

T2INT:
        CPL     P1.0                    //将测试口取反

//      ANL     IE2,    #0FBH           //若需要手动清除中断标志,可先关闭中断,
                                        //此时系统会自动清除内部的中断标志
//      ORL     IE2,    #04H            //然后再开中断即可

        RETI

;--------------------------------------

        END
```

STC MCU Limited

7.5.3 定时器2对系统时钟或外部引脚T2的时钟输入进行可编程分频输出

定时器/计数器2的原理框图如下：



定时器/计数器2的工作模式: 16位自动重装

STC创新设计，请不要抄袭，再抄袭就很无耻了

定时器/计数器2除可当定时器/计数器使用外，还可作可编程时钟输出。当定时器/计数器2用作可编程时钟输出时，不要允许相应的定时器中断, 免得CPU反复进中断。

当T2CLKO/INT_CLKO.2=1时，P3.0管脚配置为定时器2的时钟输出T2CLKO。

输出时钟频率 = T2 溢出率/2

如果T2_C/$\overline{T}$=0，定时器/计数器T2对内部系统时钟计数，则：

T2工作在1T模式(AUXR.2/T2x12=1)时的输出**时钟频率** = (SYSclk)/(65536-[RL_TH2, RL_TL2])/2

T2工作在12T模式(AUXR.2/T2x12=0)时的输出**时钟频率**=(SYSclk)/12/(65536-[RL_TH2, RL_TL2])/2

如果T2_C/$\overline{T}$=1，定时器/计数器T2是对外部脉冲输入(P3.1/T2)计数，则：

输出**时钟频率** = (T2_Pin_CLK) / (65536-[RL_TH2, RL_TL2]) / 2

上面所有的式子中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。

下面是定时器2对内部系统时钟或外部引脚T2/P3.1的时钟输入进行可编程时钟分频输出的程序举例(C和汇编):

1. C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2  系列  定时器2的可编程时钟分频输出举例----------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define FOSC 18432000L

//-----------------------------------------------

sfr         AUXR            =       0x8e;                   //辅助特殊功能寄存器
sfr         INT_CLKO        =       0x8f;                   //唤醒和时钟输出功能寄存器
sfr         T2H             =       0xD6;                   //定时器2高8位
sfr         T2L             =       0xD7;                   //定时器2低8位

sbit        T2CLKO          =       P3^0;                   //定时器2的时钟输出脚

#define     F38_4KHz        (65536-FOSC/2/38400)            //1T模式
//#define   F38_4KHz        (65536-FOSC/2/12/38400)         //12T模式

//-----------------------------------------------

void main()
{
            AUXR    |=      0x04;                           //定时器2为1T模式
//          AUXR    &=      ~0x04;                          //定时器2为12T模式
```

```
        AUXR    &=      ~0x08;          //T2_C/T=0, 对内部时钟进行时钟输出
//      AUXR    |=      0x08;           //T2_C/T=1, 对T2(P3.1)引脚的外部时钟进行时钟输出

        T2L     =       F38_4KHz;       /初始化计时值
        T2H     =       F38_4KHz >> 8;

        AUXR    |=      0x10;           //定时器2开始计时
        INT_CLKO =      0x04;           //使能定时器2的时钟输出功能

        while (1);                      //程序终止
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2  系列  定时器2可编程时钟分频输出举例-------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR            DATA    08EH            //辅助特殊功能寄存器
INT_CLKO        DATA    08FH            //唤醒和时钟输出功能寄存器
T2H             DATA    0D6H            //定时器2高8位
T2L             DATA    0D7H            //定时器2低8位

T2CLKO          BIT     P3.0            //定时器2的时钟输出脚

F38_4KHz        EQU     0FF10H          //38.4KHz(1T模式下，65536-18432000/2/38400)
//F38_4KHz      EQU     0FFECH          //38.4KHz(12T模式下,(65536-18432000/2/12/38400)

//-----------------------------------------------
```

```
        ORG     0000H
        LJMP    MAIN                                    //复位入口

//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #04H                            //定时器2为1T模式
//      ANL     AUXR,   #0FBH                           //定时器2为12T模式

        ANL     AUXR,   #0F7H           //T2_C/T=0, 对内部时钟进行时钟输出
//      ORL     AUXR,   #08H            //T2_C/T=1, 对T2(P3.1)引脚的外部时钟进行时钟输出

        MOV     T2L,    #LOW F38_4KHz                   //初始化计时值
        MOV     T2H,    #HIGH F38_4KHz
        ORL     AUXR,   #10H                            //定时器2开始计时
        MOV     INT_CLKO,       #04H                    //使能定时器2的时钟输出功能

        SJMP    $                                       //程序终止

;-----------------------------------------

        END
```

STC MCU Limited

7.5.4 定时器/计数器2作串行口波特率发生器及测试程序(C和汇编)

定时器/计数器2除可当定时器/计数器和可编程时钟输出使用外，还可作串行口波特率发生器。串行口1优先选择定时器2作为其波特率发生器，串行口2只能选择定时器2作为其波特率发生器，串行口3/串口4默认选择定时器2作为其波特率发生器。

串行口1如果工作在模式1（8位UART，波特率可变）和模式3（9位UART，波特率可变）时，其可变的波特率可以由定时器T2产生。此时：

串行口1的波特率=(定时器T2的溢出率)/4，　　注意：此时波特率也与SMOD无关。

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器T2的溢出率 = SYSclk / (65536 - [RL_TH2, RL_TL2])；

即此时，串行口1的波特率=SYSclk / (65536 - [[RL_TH2, RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2, RL_TL2])；

即此时，串行口1的波特率=SYSclk / 12 / (65536 - [RL_TH2, RL_TL2]) / 4

串行口2的工作模式只有两种：模式0（8位UART，波特率可变）和模式1（9位UART，波特率可变）。串行口2只能选择定时器T2作其波特率发生器。串行口2的波特率按如下公式计算：

串行口2的波特率=(定时器T2的溢出率)/4

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率 = SYSclk / (65536 - [RL_TH2, RL_TL2])；

即此时，串行口2的波特率=SYSclk / (65536 - [RL_TH2, RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2, RL_TL2])；

即此时，串行口2的波特率=SYSclk / 12 / (65536 - [RL_TH2, RL_TL2]) / 4

串行口3的工作模式只有两种：模式0（8位UART，波特率可变）和模式1（9位UART，波特率可变）。串行口3可以选择定时器T3作为其波特率发生器，也可以选择定时器T2作其波特率发生器。当选择定时器2作为其波特率发生器时，串行口3的波特率按如下公式计算：

串行口3的波特率=(定时器T2的溢出率)/4

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率 = SYSclk / (65536 - [RL_TH2, RL_TL2])；

即此时，串行口3的波特率=SYSclk / (65536 - [RL_TH2, RL_TL2]]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [[RL_TH2, RL_TL2])；

即此时，串行口3的波特率=SYSclk / 12 / (65536 - [[RL_TH2, RL_TL2]) / 4

串行口4的工作模式只有两种：模式0（8位UART，波特率可变）和模式1（9位UART，波特率可变）。串行口4可以选择定时器T4作为其波特率发生器，也可以选择定时器T2作其波特率发生器。当选择定时器2作为其波特率发生器时，串行口4的波特率按如下公式计算：

串行口4的波特率=(定时器T2的溢出率)/4

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率 = SYSclk / (65536 - [RL_TH2, RL_TL2])；

即此时，串行口4的波特率=SYSclk / (65536 - [RL_TH2, RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2, RL_TL2])；

即此时，串行口4的波特率=SYSclk / 12 / (65536 - [RL_TH2, RL_TL2]) / 4

上面所有的式子中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。

STC MCU Limited

7.5.4.1 定时器/计数器2作串行口1波特率发生器的测试程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器2用作串口1的波特率发生器举例---------------------- */
/* --- 研发顾问QQ：800003751----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef unsigned char        BYTE;
typedef unsigned int         WORD;

#define   FOSC    18432000L                          //系统频率
#define   BAUD    115200                             //串口波特率

#define   NONE_PARITY       0                        //无校验
#define   ODD_PARITY        1                        //奇校验
#define   EVEN_PARITY       2                        //偶校验
#define   MARK_PARITY       3                        //标记校验
#define   SPACE_PARITY      4                        //空白校验

#define PARITYBIT EVEN_PARITY                        //定义校验位

sfr       AUXR    =         0x8e;                    //辅助寄存器
sfr       T2H     =         0xd6;                    //定时器2高8位
sfr       T2L     =         0xd7;                    //定时器2低8位

sbit      P22     =         P2^2;

bit busy;

void SendData(BYTE dat);
void SendString(char *s);
void main()
{
#if (PARITYBIT == NONE_PARITY)
```

STC MCU Limited

```
        SCON    =       0x50;                                   //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        SCON    =       0xda;                                   //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        SCON    =       0xd2;                                   //9位可变波特率,校验位初始为0
#endif

        T2L     =       (65536 - (FOSC/4/BAUD));                //设置波特率重装值
        T2H     =       (65536 - (FOSC/4/BAUD))>>8;
        AUXR    =       0x14;                                   //T2为1T模式, 并启动定时器2
        AUXR    |=      0x01;                                   //选择定时器2为串口1的波特率发生器
        ES      =       1;                                      //使能串口1中断
        EA      =       1;

        SendString("STC15F2K60S2\r\nUart Test !\r\n");
        while(1);
}

/*----------------------------
UART 中断服务程序
-----------------------------*/
void Uart() interrupt 4 using 1
{
        if (RI)
        {
                RI = 0;                                         //清除RI位
                P0 = SBUF;                                      //P0显示串口数据
                P22 = RB8;                                      //P2.2显示校验位
        }
        if (TI)
        {
                TI = 0;                                         //清除TI位
                busy = 0;                                       //清忙标志
        }
}

/*----------------------------
发送串口数据
-----------------------------*/
void SendData(BYTE dat)
{
        while (busy);                                           //等待前面的数据发送完成
        ACC = dat;                                              //获取校验位P (PSW.0)
        if (P)                                                  //根据P来设置校验位
        {
        #if (PARITYBIT == ODD_PARITY)
```

STC MCU Limited

```
            TB8 = 0;                    //设置校验位为0
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 1;                    //设置校验位为1
        #endif
        }
        else
        {
        #if (PARITYBIT == ODD_PARITY)
            TB8 = 1;                    //设置校验位为1
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 0;                    //设置校验位为0
        #endif
         }
        busy = 1;
        SBUF = ACC;                     //写数据到UART数据寄存器
}

/*---------------------------
发送字符串
---------------------------*/
void SendString(char *s)
{
        while (*s)                      //检测字符串结束标志
        {
            SendData(*s++);             //发送当前字符
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ------------------------------------------------------------------------------ */
/* --- STC15F2K60S2 系列 定时器2用作串口1的波特率发生器举例---------------------------------------- */
/* --- 研发顾问QQ：800003751-------------------------------------------------------------------------- */
/* --- Fax: 86-755-82905966 -------------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 -------------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com --------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                   */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                   */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#define   NONE_PARITY   0                       //无校验
#define   ODD_PARITY    1                       //奇校验
#define   EVEN_PARITY   2                       //偶校验
#define   MARK_PARITY   3                       //标记校验
#define   SPACE_PARITY  4                       //空白校验

#define   PARITYBIT     EVEN_PARITY             //定义校验位

//-----------------------------------------

AUXR    EQU     08EH                            //辅助寄存器
T2H     DATA    0D6H                            //定时器2高8位
T2L     DATA    0D7H                            //定时器2低8位

//-----------------------------------------
BUSY    BIT     20H.0                           //忙标志位
//-----------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     0023H
        LJMP    UART_ISR
//-----------------------------------------
        ORG     0100H
MAIN:
        CLR     BUSY
        CLR     EA
        MOV     SP,     #3FH

#if (PARITYBIT == NONE_PARITY)
        MOV     SCON,   #50H                    //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
```

STC MCU Limited

```
        MOV     SCON,   #0DAH                   //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     SCON,   #0D2H                   //9位可变波特率,校验位初始为0
#endif

//-------------------------------
        MOV     T2L,    #0D8H                   //设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H                    //T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H                    //选择定时器2为串口1的波特率发生器
         SETB   ES                              //使能串口中断
        SETB    EA

        MOV     DPTR,   #TESTSTR                //发送测试字符串
        LCALL   SENDSTRING

        SJMP    $
;------------------------------------------
TESTSTR:
        DB  "STC15F2K60S2 Uart1 Test !",0DH,0AH,0

;/*---------------------------
;UART 中断服务程序
;---------------------------*/
UART_ISR:
        PUSH    ACC
        PUSH    PSW
        JNB     RI,     CHECKTI                 //检测RI位
        CLR     RI                              //清除RI位
        MOV     P0,     SBUF                    //P0显示串口数据
        MOV     C,      RB8
        MOV     P2.2,   C                       //P2.2显示校验位
CHECKTI:
        JNB     TI,     ISR_EXIT                //检测TI位
        CLR     TI                              //清除TI位
        CLR     BUSY                            //清忙标志
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

;/*---------------------------
;发送串口数据
;---------------------------*/
SENDDATA:
        JB      BUSY,   $                       //等待前面的数据发送完成
        MOV     ACC,    A                       //获取校验位P (PSW.0)
        JNB     P,      EVEN1INACC              //根据P来设置校验位
```

STC MCU Limited

```
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        CLR     TB8                     //设置校验位为0
#elif (PARITYBIT == EVEN_PARITY)
        SETB    TB8                     //设置校验位为1
#endif
        SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        SETB    TB8                     //设置校验位为1
#elif (PARITYBIT == EVEN_PARITY)
        CLR     TB8                     //设置校验位为0
#endif
PARITYBITOK:                            //校验位设置完成
        SETB    BUSY
        MOV     SBUF,   A               //写数据到UART数据寄存器
        RET

;/*---------------------------
;发送字符串
//---------------------------*/
SENDSTRING:
        CLR     A
        MOVC    A,      @A+DPTR         //读取字符
        JZ      STRINGEND               //检测字符串结束标志
        INC     DPTR                    //字符串地址+1
        LCALL   SENDDATA                //发送当前字符
        SJMP    SENDSTRING
STRINGEND:
        RET
//-----------------------------------------
        END
```

STC MCU Limited

7.5.4.2 定时器/计数器2作串行口2波特率发生器的测试程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器2用作串口2的波特率发生器举例---------------------- */
/* --- 研发顾问QQ：800003751----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef unsigned char       BYTE;
typedef unsigned int        WORD;

#define   FOSC    18432000L                 //系统频率
#define   BAUD    115200                    //串口波特率
#define   TM      (65536 - (FOSC/4/BAUD))

#define   NONE_PARITY     0                 //无校验
#define   ODD_PARITY      1                 //奇校验
#define   EVEN_PARITY     2                 //偶校验
#define   MARK_PARITY     3                 //标记校验
#define   SPACE_PARITY    4                 //空白校验

#define PARITYBIT EVEN_PARITY               //定义校验位

sfr       AUXR    =       0x8e;             //辅助寄存器
sfr       S2CON   =       0x9a;             //UART2 控制寄存器
sfr       S2BUF   =       0x9b;             //UART2 数据寄存器
sfr       T2H     =       0xd6;             //定时器2高8位
sfr       T2L     =       0xd7;             //定时器2低8位
sfr       IE2     =       0xaf;             //中断控制寄存器2

#define   S2RI    0x01                      //S2CON.0
#define   S2TI    0x02                      //S2CON.1
```

STC MCU Limited

```
#define   S2RB8   0x04                  //S2CON.2
#define   S2TB8   0x08                  //S2CON.3

bit       busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
        S2CON = 0x50;                           //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        S2CON = 0xda;                           //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        S2CON = 0xd2;                           //9位可变波特率,校验位初始为0
#endif

        T2L = TM;                               //设置波特率重装值
        T2H = TM>>8;
        AUXR = 0x14;                            //T2为1T模式, 并启动定时器2
        IE2 = 0x01;                             //使能串口2中断
        EA = 1;

        SendString("STC15F2K60S2\r\nUart2 Test !\r\n");
        while(1);
}

/*----------------------------
UART2 中断服务程序
----------------------------*/
void Uart2() interrupt 8 using 1
{
        if (S2CON & S2RI)
        {
                S2CON &= ~S2RI;                 //清除S2RI位
                P0 = S2BUF;                     //P0显示串口数据
                P2 = (S2CON & S2RB8);           //P2.2显示校验位
        }
        if (S2CON & S2TI)
        {
                S2CON &= ~S2TI;                 //清除S2TI位
                busy = 0;                       //清忙标志
        }
}

/*----------------------------
```

STC MCU Limited

```
发送串口数据
-----------------------------*/
void SendData(BYTE dat)
{
        while (busy);                                   //等待前面的数据发送完成
        ACC = dat;                                      //获取校验位P (PSW.0)
        if (P)                                          //根据P来设置校验位
        {
        #if (PARITYBIT == ODD_PARITY)
                S2CON &= ~S2TB8;                        //设置校验位为0
        #elif (PARITYBIT == EVEN_PARITY)
                S2CON |= S2TB8;                         //设置校验位为1
        #endif
        }
        else
        {
        #if (PARITYBIT == ODD_PARITY)
                S2CON |= S2TB8;                         //设置校验位为1
        #elif (PARITYBIT == EVEN_PARITY)
                S2CON &= ~S2TB8;                        //设置校验位为0
        #endif
        }
        busy = 1;
        S2BUF = ACC;                                    //写数据到UART2数据寄存器
}

/*---------------------------
发送字符串
---------------------------*/
void SendString(char *s)
{
        while (*s)                                      //检测字符串结束标志
        {
                SendData(*s++);                         //发送当前字符
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*------------------------------------------------------------------------------------------------------ */
/* --- STC MCU  Limited. ------------------------------------------------------------------------------ */
/* --- STC15F2K60S2 系列 定时器2用作串口2的波特率发生器举例---------------------- */
/* --- 研发顾问QQ：800003751------------------------------------------------------------------------------ */
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com ------------------------------------------------------------------------------ */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*------------------------------------------------------------------------------------------------------ */

//假定测试芯片的工作频率为18.432MHz

#define   NONE_PARITY          0          //无校验
#define   ODD_PARITY           1          //奇校验
#define   EVEN_PARITY          2          //偶校验
#define   MARK_PARITY          3          //标记校验
#define   SPACE_PARITY         4          //空白校验

#define   PARITYBIT EVEN_PARITY           //定义校验位

//-----------------------------------------

AUXR    EQU     08EH            //辅助寄存器
S2CON   EQU     09AH            //UART2 控制寄存器
S2BUF   EQU     09BH            //UART2 数据寄存器
T2H     DATA    0D6H            //定时器2高8位
T2L     DATA    0D7H            //定时器2低8位
IE2     EQU     0AFH            //中断控制寄存器2

S2RI    EQU     01H             //S2CON.0
S2TI    EQU     02H             //S2CON.1
S2RB8   EQU     04H             //S2CON.2
S2TB8   EQU     08H             //S2CON.3
//-----------------------------------------
BUSY    BIT     20H.0           //忙标志位
//-----------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     0043H
        LJMP    UART2_ISR
//-----------------------------------------
        ORG     0100H
MAIN:
        CLR     BUSY
```

STC MCU Limited

```
        CLR     EA
        MOV     SP,     #3FH
#if (PARITYBIT == NONE_PARITY)
        MOV     S2CON,  #50H                    //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        MOV     S2CON,  #0DAH                   //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     S2CON,  #0D2H                   //9位可变波特率,校验位初始为0
#endif

//---------------------------------
        MOV     T2L,    #0D8H                   //设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H                    //T2为1T模式, 并启动定时器2
        ORL     IE2,    #01H                    //使能串口2中断
        SETB    EA

        MOV     DPTR,   #TESTSTR                //发送测试字符串
        LCALL   SENDSTRING

        SJMP    $
;----------------------------------------
TESTSTR:
        DB  "STC15F2K60S2 Uart2 Test !",0DH,0AH,0

;/*-----------------------------
;UART2 中断服务程序
;-----------------------------*/
UART2_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     A,      S2CON                   ;读取UART2控制寄存器
        JNB     ACC.0,  CHECKTI                 ;检测S2RI位
        ANL     S2CON,  #NOT S2RI               ;清除S2RI位
         MOV    P0,     S2BUF                   ;P0显示串口数据
        ANL     A,      #S2RB8                  ;
        MOV      P2,    A                       ;P2.2显示校验位
CHECKTI:                        ;
        MOV     A,      S2CON                   ;读取UART2控制寄存器
        JNB     ACC.1,  ISR_EXIT                ;检测S2TI位
        ANL     S2CON,  #NOT S2TI               ;清除S2TI位
        CLR     BUSY                            ;清忙标志
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI
```

STC MCU Limited

```
;/*------------------------------
;发送串口数据
;------------------------------*/
SENDDATA:
        JB      BUSY,   $                       //等待前面的数据发送完成
        MOV     ACC,    A                       //获取校验位P (PSW.0)
        JNB     P,      EVEN1INACC              //根据P来设置校验位
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        ANL     S2CON,  #NOT S2TB8              //设置校验位为0
#elif (PARITYBIT == EVEN_PARITY)
        ORL     S2CON,  #S2TB8                  //设置校验位为1
#endif
        SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        ORL     S2CON,  #S2TB8                  //设置校验位为1
#elif (PARITYBIT == EVEN_PARITY)
        ANL     S2CON,  #NOT S2TB8              //设置校验位为0
#endif
PARITYBITOK:                                    //校验位设置完成
        SETB    BUSY
        MOV     S2BUF,  A                       //写数据到UART2数据寄存器
        RET

;/*------------------------------
;发送字符串
//------------------------------*/
SENDSTRING:
        CLR     A
        MOVC    A,      @A+DPTR                 //读取字符
        JZ      STRINGEND                       //检测字符串结束标志
        INC     DPTR                            //字符串地址+1
        LCALL   SENDDATA                        //发送当前字符
        SJMP    SENDSTRING
STRINGEND:
        RET
//-----------------------------------------
        END
```

STC MCU Limited

7.6 定时器/计数器3及定时器/计数器4

STC15W4K60S4还新增了两个16位定时/计数器：T3和T4。T3、T4和T2一样，它们的工作模式固定为16位自动重装载模式。T3和T4既可以当定时器/计数器用，也可以当可编程时钟输出和串口的波特率发生器。

下面首先介绍与定时器/计数器T3和T4相关的寄存器：

7.6.1 定时器/计数器3和定时器/计数器4的相关特殊功能寄存器

与定时器/计数器3和定时器/计数器4有关的特殊功能寄存器：

符号	描述	地址	位地址及其符号 MSB LSB								复位值
T4T3M	T4和T3的控制寄存器	D1H	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO	0000 0000B
T4H	定时器4高8位寄存器	D2H									0000 0000B
T4L	定时器4低8位寄存器	D3H									0000 0000B
T3H	定时器3高8位寄存器	D4H									0000 0000B
T3L	定时器3低8位寄存器	D5H									0000 0000B
IE2	Interrupt Enable register	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B

STC MCU Limited

1、定时器T4和T3的控制寄存器：T4T3M（地址：0xD1）

T4T3M（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO

B7 - T4R：定时器4运行控制位。
　　0：不允许定时器4运行；
　　1：允许定时器4运行。

B6 - T4_C/$\overline{T}$：控制定时器4用作定时器或计数器。
　　0，用作定时器（对内部系统时钟进行计数）；
　　1，用作计数器（对引脚T4/P0.7的外部脉冲进行计数）

B5 - T4x12：定时器4速度控制位。
　　0：定时器4速度是8051单片机定时器的速度，即12分频；
　　1：定时器4速度是8051单片机定时器速度的12倍，即不分频。

B4 - T4CLKO：是否允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

1：允许将P0.6脚配置为定时器4的时钟输出T4CLKO，输出时钟频率=T4溢出率/2

如果T4_C/$\overline{T}$=0，定时器/计数器T4是对内部系统时钟计数，则：

T4工作在1T模式(T4T3M.5/T4x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH4, RL_TL4])/2

T4工作在12T模式(T4T3M.5/T4x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH4, RL_TL4])/2

如果T4_C/$\overline{T}$=1，定时器/计数器T4是对外部脉冲输入(P0.7/T4)计数，则：

输出时钟频率 = (T4_Pin_CLK) / (65536-[RL_TH4, RL_TL4])/2

0：不允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

B3 - T3R：定时器3运行控制位。

0：不允许定时器3运行；

1：允许定时器3运行。

B2 - T3_C/$\overline{T}$：控制定时器3用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T3/P0.5的外部脉冲进行计数)

B1 - T3x12：定时器3速度控制位。

0：定时器3速度是8051单片机定时器的速度，即12分频；

1：定时器3速度是8051单片机定时器速度的12倍，即不分频。

B0 - T3CLKO：是否允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

1：允许将P0.4脚配置为定时器3的时钟输出T3CLKO，输出时钟频率=T3溢出率/2

如果T3_C/$\overline{T}$=0，定时器/计数器T3是对内部系统时钟计数，则：

T3工作在1T模式(T4T3M.1/T3x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH3, RL_TL3])/2

T3工作在12T模式(T4T3M.1/T3x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH3, RL_TL3])/2

如果T3_C/$\overline{T}$=1，定时器/计数器T3是对外部脉冲输入(P0.5/T3)计数，则：

输出时钟频率 = (T3_Pin_CLK) / (65536-[RL_TH3, RL_TL3])/2

0：不允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

2、定时器T3和T4的中断控制寄存器:IE2

IE2：中断允许寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4：定时器4的中断允许位。

1，允许定时器4产生中断；

0，禁止定时器4产生中断。

ET3：定时器3的中断允许位。

1，允许定时器3产生中断；

0，禁止定时器3产生中断。

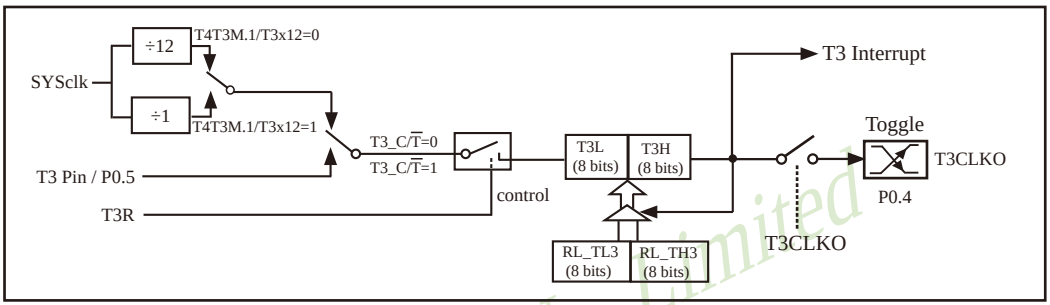
STC MCU Limited

7.6.2 定时器/计数器3的应用(STC创新设计，请不要抄袭)

定时器/计数器3既可以当定时器/计数器用，也可以当可编程时钟输出和串口的波特率发生器。

7.6.2.1 定时器/计数器3作定时器

定时器/计数器3的原理框图如下：



定时器/计数器3的工作模式: 16位自动重装载

STC创新设计，请不要抄袭，再抄袭就很无耻了

T3R/T4T3M.3为T4T3M寄存器内的控制位，T4T3M寄存器各位的具体功能描述见上节T4T3M寄存器的介绍。

当T3_C/$\overline{T}$=0时，多路开关连接到系统时钟输出，T3对内部系统时钟计数，T3工作在定时方式。当T3_C/$\overline{T}$=1时，多路开关连接到外部脉冲输入P0.5/T3，即T3工作在计数方式。

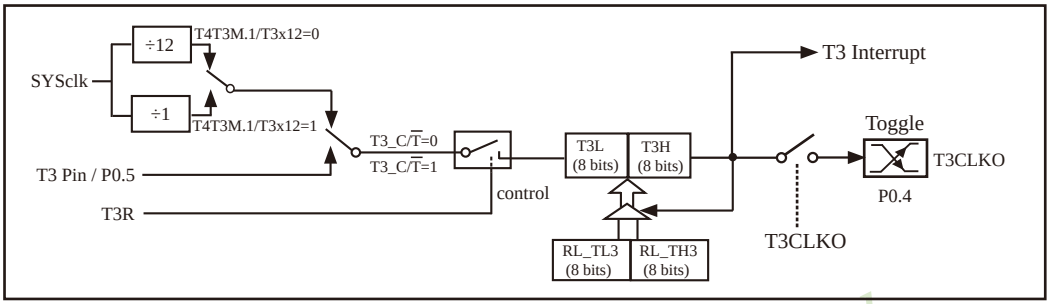
STC15W4K60S4系列单片机的定时器3有两种计数速率：一种是12T模式，每12个时钟加1，与传统8051单片机相同；另外一种是1T模式，每个时钟加1，速度是传统8051单片机的12倍。T3的速率由特殊功能寄存器T4T3M中的T3x12决定，如果T3x12=0，T3则工作在12T模式；如果T3x12=1，T3则工作在1T模式。

定时器3有2个隐藏的寄存器RL_TH3和RL_TL3。RL_TH3与T3H共有同一个地址，RL_TL3与T3L共有同一个地址。当T3R=0即定时器/计数器3被禁止工作时，对T3L写入的内容会同时写入RL_TL3，对T3H写入的内容也会同时写入RL_TH3。当T3R=1即定时器/计数器3被允许工作时，对T3L写入内容，实际上不是写入当前寄存器T3L中,而是写入隐藏的寄存器RL_TL3中；对T3H写入内容，实际上也不是写入当前寄存器T3H中，而是写入隐藏的寄存器RL_TH3。当读T3H和T3L的内容时，所读的内容就是T3H和T3L的内容，而不是RL_TH3和RL_TL3的内容。

这样可以巧妙地实现16位重装载定时器。[T3L,T3H]的溢出不仅置位被隐藏的中断请求标志位(定时器3的中断请求标志位对用户不可见)，使CPU转去执行定时器3中断的程序，而且会自动将[RL_TL3,RL_TH3]的内容重新装入[T3L,T3H]。

7.6.2.2 定时器/计数器3对系统时钟或外部引脚T3的时钟输入进行可编程时钟分频输出

定时器/计数器3的原理框图如下：



定时器/计数器3的工作模式: 16位自动重装载

STC创新设计，请不要抄袭，再抄袭就很无耻了

定时器/计数器3除可当定时器/计数器使用外，还可作可编程时钟输出。当定时器/计数器3用作可编程时钟输出时，不要允许相应的定时器中断, 免得CPU反复进中断。

当T3CLKO/T4T3M.0=1时，P0.4管脚配置为定时器3的时钟输出T3CLKO。

输出时钟频率 = T3 溢出率/2

如果T3_C/$\overline{T}$=0，定时器/计数器T3对内部系统时钟计数，则：

T3工作在1T模式(T4T3M.1/T3x12=1)时的输出时钟频率 = (SYSclk)/(65536-[RL_TH3, RL_TL3])/2

T3工作在12T模式(T4T3M.1/T3x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH3, RL_TL3])/2

如果T3_C/$\overline{T}$=1，定时器/计数器T3是对外部脉冲输入(P0.5/T3)计数，则：

输出时钟频率 = (T3_Pin_CLK) / (65536-[RL_TH3, RL_TL3])/2

上面所有的式子中RL_TH3是T3H的重装载寄存器，RL_TL3是T3L的重装载寄存器。

7.6.2.3 定时器/计数器3作串行口3的波特率发生器

定时器/计数器3除可当定时器/计数器和可编程时钟输出使用外，还可作串行口3波特率发生器。串行口3默认选择定时器2作为其波特率发生器，但通过设置S3ST3/S3CON.6，串行口3也可以选择定时器3作为其波特率发生器。

S3CON：串行口3控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	name	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3ST3：串口3(UART3)选择定时器3作波特率发生器的控制位

0，选择定时器2作为串口3(UART3)的波特率发生器；

1，选择定时器3作为串口3(UART3)的波特率发生器

串行口3的工作模式只有两种：模式0（8位UART，波特率可变）和模式1（9位UART，波特率可变）。当串行口3被设置为选择定时器3作为其波特率发生器时，串行口3的波特率按如下公式计算：

串行口3的波特率=(定时器T3的溢出率)/4

当T3工作在1T模式(T4T3M.1/T3x12=1)时，定时器3的溢出率 = SYSclk / (65536 - [RL_TH3,RL_TL3])；

即此时，串行口3的波特率=SYSclk / (65536 - [RL_TH3, RL_TL3]) / 4

当T3工作在12T模式(T4T3M.1/T3x12=0)时，定时器3的溢出率=SYSclk /12/(65536 - [RL_TH3, RL_TL3])；

即此时，串行口3的波特率=SYSclk / 12 / (65536 - [RL_TH3, RL_TL3]) / 4

上面所有的式子中RL_TH3是T3H的重装载寄存器，RL_TL3是T3L的重装载寄存器。

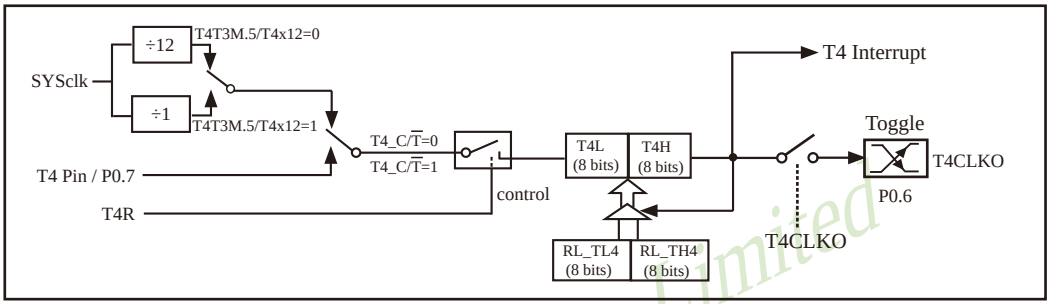
STC MCU Limited

7.6.3 定时器/计数器4的应用(STC创新设计，请不要抄袭)

定时器/计数器4**既可以当定时器/计数器用，也可以当可编程时钟输出和串口的波特率发**生器。

7.6.3.1 定时器/计数器4作定时器

定时器/计数器4的原理框图如下：



定时器/计数器4的工作模式: 16位自动重装载

STC创新设计，请不要抄袭，再抄袭就很无耻了

T4R/T4T3M.7为T4T3M寄存器内的控制位，T4T3M寄存器各位的具体功能描述见上节**T4T3M寄存器的介绍。**

当T4_C/$\overline{T}$=0时，多路开关连接到系统时钟输出，T4对内部系统时钟计数，T4工作在定时方式。当T4_C/$\overline{T}$=1时，多路开关连接到外部脉冲输入P0.7/T4，即T4工作在计数方式。

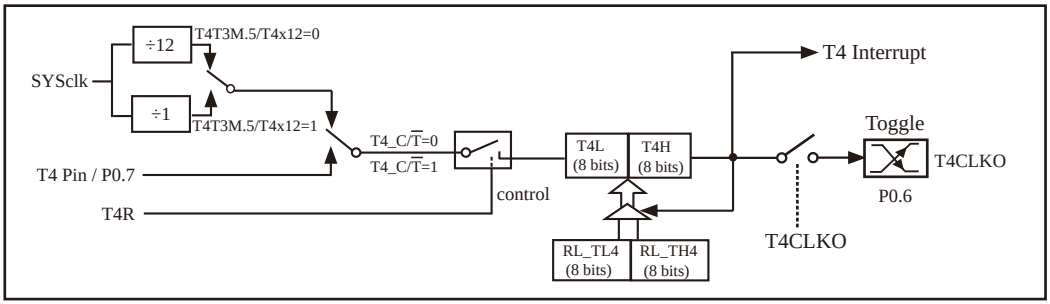
STC15W4K60S4**系列单片机的定时器4有两种计数速率：一种是12T模式，每12个时钟加1，**与传统8051**单片机相同；另外一种是1T模式，每个时钟加1，速度是传统8051单片机的12倍。**T4**的速率由特殊功能寄存器T4T3M中的T4x12决定，如果T4x12=0，T4则工作在12T模式；如果**T4x12=1，T4则工作在1T模式。**

定时器4有2**个隐藏的寄存器**RL_TH4和RL_TL4。RL_TH4与T4H**共有同一个地址，**RL_TL4与T4L共有同一个地址。当T4R=0即定时器/计数器4被禁止工作时，对T4L写入的内容会同时写入RL_TL4，对T4H写入的内容也会同时写入RL_TH4。当T4R=1即定时器/计数器4被允许工作时，对T4L写入内容，实际上不是写入当前寄存器T4L中,而是写入隐藏的寄存器RL_TL4中；对T4H写入内容，实际上也不是写入当前寄存器T4H中，而是写入隐藏的寄存器RL_TH4。当读T4H和T4L的内容时，所读的内容就是T4H和T4L的内容，而不是RL_TH4和RL_TL4的内容。

这样可以巧妙地实现16位重装载定时器。[T4L,T4H]**的溢出不仅置位被隐藏的**中断请求标志位(定时器4的中断请求标志位对用户不可见)，使CPU转去执行定时器4中断的程序，而且会自动将[RL_TL4,RL_TH4]的内容重新装入[T4L,T4H]。

7.6.3.2 定时器/计数器4对系统时钟或外部引脚T4的时钟输入进行可编程时钟分频输出

定时器/计数器4的原理框图如下：



定时器/计数器4的工作模式: 16位自动重装载

STC创新设计，请不要抄袭，再抄袭就很无耻了

定时器/计数器4除可当定时器/计数器使用外，还可作可编程时钟输出。当定时器/计数器4用作可编程时钟输出时，不要允许相应的定时器中断, 免得CPU反复进中断。

当T4CLKO/T4T3M.4=1时，P0.6管脚配置为定时器4的时钟输出T4CLKO。

输出时钟频率 = T4 溢出率/2

如果T4_C/$\overline{T}$=0，定时器/计数器T4对内部系统时钟计数，则：

T4工作在1T模式(T4T3M.5/T4x12=1)时的输出时钟频率 = (SYSclk)/(65536-[RL_TH4, RL_TL4])/2

T4工作在12T模式(T4T3M.5/T4x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH4, RL_TL4])/2

如果T4_C/$\overline{T}$=1，定时器/计数器T4是对外部脉冲输入(P0.7/T4)计数，则：

输出时钟频率 = (T4_Pin_CLK) / (65536-[RL_TH4, RL_TL4])/2

上面所有的式子中RL_TH4是T4H的重装载寄存器，RL_TL4是T4L的重装载寄存器。

7.6.3.3 定时器/计数器4作串行口4的波特率发生器

定时器/计数器4除可当定时器/计数器和可编程时钟输出使用外，还可作串行口4波特率发生器。串行口4默认选择定时器2作为其波特率发生器，但通过设置S4ST4/S4CON.6，串行口4也可以选择定时器4作为其波特率发生器。

S4CON：串行口4控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	name	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4ST4：串口4(UART4)选择定时器4作波特率发生器的控制位

0，选择定时器2作为串口4(UART4)的波特率发生器；

1，选择定时器4作为串口4(UART4)的波特率发生器

串行口4的工作模式只有两种：模式0（8位UART，波特率可变）和模式1（9位UART，波特率可变）。当串行口4被设置为选择定时器4作为其波特率发生器时，串行口4的波特率按如下公式计算：

串行口4的波特率=(定时器T4的溢出率)/4

当T4工作在1T模式(T4T3M.5/T4x12=1)时，定时器4的溢出率 = SYSclk / (65536 - [RL_TH4,RL_TL4])；

即此时，串行口4的波特率=SYSclk / (65536 - [RL_TH4, RL_TL4]) / 4

当T4工作在12T模式(T4T3M.5/T4x12=0)时，定时器4的溢出率=SYSclk /12/(65536 - [RL_TH4, RL_TL4])；

即此时，串行口4的波特率=SYSclk / 12 / (65536 - [RL_TH4, RL_TL4]) / 4

上面所有的式子中RL_TH4是T4H的重装载寄存器，RL_TL4是T4L的重装载寄存器。

STC MCU Limited

7.7 如何将定时器T0/T1/T2/T3/T4的速度提高12倍

1、定时器T0/T1/T2的速度控制寄存器位：T0x12/T1x12/T2x12

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2

定时器0、定时器1和定时器2:

STC15W4K60S4系列是1T的8051单片机，为了兼容传统8051，定时器0、和定时器1和定时器2复位后是传统8051的速度，即12分频，这是为了兼容传统8051。但也可不进行12分频，实现真正的1T。

T0x12：定时器0速度控制位

0，定时器0是传统8051速度，12分频；

1，定时器0的速度是传统8051的12倍，不分频

T1x12：定时器1速度控制位

0，定时器1是传统8051速度，12分频；

1，定时器1的速度是传统8051的12倍，不分频

如果串口1用定时器1做波特率发生器，T1x12位就可以控制UART异步串口是12T还是1T了。

T2x12：定时器2速度控制位

0，定时器2是传统8051速度，12分频；

1，定时器2的速度是传统8051的12倍，不分频

如果串口1或串口2用T2作为波特率发生器，则由T2x12决定串口1或串口2是12T

串口1的模式0:

STC15W4K60S4系列是1T的8051单片机,为了兼容传统8051,串口1复位后是兼容传统8051的

UART_M0x6：串口1模式0的通信速度设置位。

0，串口1模式0的速度是传统8051单片机串口的速度，12分频；

1，串口1模式0的速度是传统8051单片机串口速度的6倍，2分频

如果用定时器T1做波特率发生器时,串口1的速度由T1的溢出率决定

T2R：定时器2允许控制位

0，不允许定时器2运行；

1，允许定时器2运行

T2_C/$\overline{T}$：控制定时器2用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T2/P3.1的外部脉冲进行计数)

EXTRAM：内部/外部RAM存取控制位

0，允许使用逻辑上在片外、物理上在片内的扩展RAM；

1，禁止使用逻辑上在片外、物理上在片内的扩展RAM

S1ST2：串口1(UART1)选择定时器2作波特率发生器的控制位

0，选择定时器1作为串口1(UART1)的波特率发生器；

1，选择定时器2作为串口1(UART1)的波特率发生器，此时定时器1得到释放，可以作为独立定时器使用

注意：有串口2的单片机，串口2永远是使用定时器2作为波特率发生器,串口2不能够选择定时器1做波特率发生器，串口1可以选择定时器1做波特率发生器，也可以选择定时器2作为波特率发生器.

STC MCU Limited

2、定时器T4和T3的速度控制寄存器位：T4x12/T3x12

T4T3M(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO

B5 - T4x12：定时器4速度控制位。

0：定时器4速度是8051单片机定时器的速度，即12分频；

1：定时器4速度是8051单片机定时器速度的12倍，即不分频。

B1 - T3x12：定时器3速度控制位。

0：定时器3速度是8051单片机定时器的速度，即12分频；

1：定时器3速度是8051单片机定时器速度的12倍，即不分频。

B7 - T4R：定时器4运行控制位。

0：不允许定时器4运行；

1：允许定时器4运行。

B6 - T4_C/$\overline{T}$：控制定时器4用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T4/P0.7的外部脉冲进行计数)

B4 - T4CLKO：是否允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

1：允许将P0.6脚配置为定时器4的时钟输出T4CLKO，输出时钟频率=T4溢出率/2

如果T4_C/$\overline{T}$=0，定时器/计数器T4是对内部系统时钟计数，则：

T4工作在1T模式(T4T3M.5/T4x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH4, RL_TL4])/2

T4工作在12T模式(T4T3M.5/T4x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH4, RL_TL4])/2

如果T4_C/$\overline{T}$=1，定时器/计数器T4是对外部脉冲输入(P0.7/T4)计数，则：

输出时钟频率 = (T4_Pin_CLK) / (65536-[RL_TH4, RL_TL4])/2

0：不允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

B3 - T3R：定时器3运行控制位。

0：不允许定时器3运行；

1：允许定时器3运行。

B2 - T3_C/$\overline{T}$：控制定时器3用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T3/P0.5的外部脉冲进行计数)

B0 - T3CLKO：是否允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

1：允许将P0.4脚配置为定时器3的时钟输出T3CLKO，输出时钟频率=T3溢出率/2

如果T3_C/$\overline{T}$=0，定时器/计数器T3是对内部系统时钟计数，则：

T3工作在1T模式(T4T3M.1/T3x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH3, RL_TL3])/2

T3工作在12T模式(T4T3M.1/T3x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH3, RL_TL3])/2

如果T3_C/$\overline{T}$=1，定时器/计数器T3是对外部脉冲输入(P0.5/T3)计数，则：

输出时钟频率 = (T3_Pin_CLK) / (65536-[RL_TH3, RL_TL3])/2

0：不允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

STC MCU Limited

7.8 可编程时钟输出(也可作分频器使用)

STC15系列单片机最多有六路可编程时钟输出(如STC15W4K60S4系列),这六路可编程时钟输出分别是: MCLKO/P5.4或MCLKO_2/XTAL2/P1.6, T0CLKO/P3.5, T1CLKO/P3.4, T2CLKO/P3.0, T3CLKO/P0.4, T4CLKO/P0.6. 由于单片机I/O口的对外输出速度最快不超过13.5MHz，所以可编程时钟对外输出速度最快也不超过13.5MHz。

STC15全系列的可编程时钟输出的类型如下表所示。

可编程时钟输出 / 单片机型号	主时钟输出(MCLKO/P5.4)	定时器/计数器0时钟输出(T0CLKO/P3.5)	定时器/计数器1时钟输出(T1CLKO/P3.4)	定时器/计数器2时钟输出(T2CLKO/P3.0)	定时器/计数器3时钟输出(T3CLKO/P.0.4)	定时器/计数器4时钟输出(T4CLKO/P0.6)
STC15F101W系列	该系列主时钟输出在MCLKO/P3.4	√		√		
STC15F408AD系列	√	√		√		
STC15W201S系列	√	√		√		
STC15W408AS系列	√ (该系列主时钟输出还可在MCLKO_2/XTAL2/P1.6)	√		√		
STC15W408S系列	√ (该系列主时钟输出还可在MCLKO_2/P1.6)	√	√	√		
STC15W1K16S系列	√ (该系列主时钟输出还可在MCLKO_2/XTAL2/P1.6)	√	√	√		
STC15F2K60S2系列	√	√	√	√		
STC15W4K60S4系列	√ (该系列主时钟输出还可在MCLKO_2/XTAL2/P1.6)	√	√	√	√	√

上表中√表示对应的系列有相应的可编程时钟输出。

特别注意：对于STC15W1K16S系列和STC15W408S单片机，若要使用T0CLKO时钟输出功能，必须将P3.5口设置为强推挽输出模式。

STC MCU Limited

7.8.1 与可编程时钟输出有关的特殊功能寄存器

符号	描述	地址	位地址及其符号 MSB LSB	复位值
AUXR	辅助寄存器	8EH	T0x12 | T1x12 | UART_M0x6 | T2R | T2_C/$\overline{T}$ | T2x12 | EXTRAM | S1ST2	0000 0001B
INT_CLKO AUXR2	External Interrupt enable and Clock output register	8FH	- | EX4 | EX3 | EX2 | - | T2CLKO | T1CLKO | T0CLKO	x000 x000B
CLK_DIV (PCON2)	时钟分配器	97H	MCKO_S1 | MCKO_S1 | ADRJ | Tx_Rx | MCLKO_2 | CLKS2 | CLKS1 | CLKS0	0000 0000B
T4T3M	T4和T3的控制寄存器	D1H	T4R | T4_C/$\overline{T}$ | T4x12 | T4CLKO | T3R | T3_C/$\overline{T}$ | T3x12 | T3CLKO	0000 0000B

特殊功能寄存器INT_CLKO/AUXR/CLK_DIV/T4T3M的C语言声明：

```
sfr     INT_CLKO    =  0x8F;    //新增加的特殊功能寄存器INT_CLKO的地址声明
sfr     AUXR        =  0x8E;    //特殊功能寄存器AUXR的地址声明
sfr     CLK_DIV     =  0x97;    //特殊功能寄存器CLK_DIV的地址声明
sfr     T4T3M       =  0xD1;    //新增加的特殊功能寄存器T4T3M的地址声明
```

特殊功能寄存器INT_CLKO/AUXR/CLK_DIV/T4T3M的汇编语言声明：

```
INT_CLKO    EQU  8FH    ;新增加的特殊功能寄存器INT_CLKO的地址声明
AUXR        EQU  8EH    ;特殊功能寄存器AUXR的地址声明
CLK_DIV     EQU  97H    ;特殊功能寄存器CLK_DIV的地址声明
T4T3M       EQU  D1H    ;新增加的特殊功能寄存器T4T3M的地址声明
```

1. CLK_DIV (PCON2) : 时钟分频寄存器(不可位寻址)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， 输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， 输出时钟频率 = MCLK / 4

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK是指主时钟频率。

STC15W4K60S4系列单片机在MCLKO/P5.4口或MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC Mcu Limited

STC15W系列单片机通过CLK_DIV.3/MCLKO_2位来选择是在MCLKO/P5.4口对外输出主时钟，还是在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位

0：在MCLKO/P5.4口对外输出主时钟；

1：在MCLKO_2/XTAL2/P1.6口对外输出主时钟；

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

ADRJ：ADC转换结果调整

0：ADC_RES[7:0]存放高8位ADC结果，ADC_RESL[1:0]存放低2位ADC结果

1：ADC_RES[1:0]存放高2位ADC结果，ADC_RESL[7:0]存放低8位ADC结果

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 （系统时钟是指对主时钟进行分频后供给CPU、串行口、SPI、定时器、CCP/PWM/PCA、A/D转换的实际工作时钟）
0	0	0	主时钟频率/1,不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

STC MCU Limited

2. INT_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

B0 – T0CLKO ： 是否允许将P3.5/T1脚配置为定时器0(T0)的时钟输出T0CLKO

1，将P3.5/T1管脚配置为定时器0的时钟输出T0CLKO，输出时钟频率=T0溢出率/2

若定时器/计数器T0工作在定时器模式0(16位自动重装载模式)时，

如果C/$\overline{T}$=0，定时器/计数器T0是对内部系统时钟计数，则：

T0工作在1T模式(AUXR.7/T0x12=1)时的输出频率 = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH0, RL_TL0])/2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

若定时器/计数器T0工作在定时器模式2(8位自动重装模式)，

如果C/$\overline{T}$=0, 定时器/计数器T0是对内部系统时钟计数，则：

T0工作在1T模式(AUXR.7/T0x12=1)时的输出频率 = (SYSclk) / (256-TH0) / 2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出频率 = (SYSclk) / 12 / (256-TH0) / 2

如果C/$\overline{T}$=1，定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (256-TH0) / 2

0，不允许P3.5/T1管脚被配置为定时器0的时钟输出

B1 – T1CLKO：是否允许将P3.4/T0脚配置为定时器1(T1)的时钟输出T1CLKO

1，将P3.4/T0管脚配置为定时器1的时钟输出T1CLKO，输出时钟频率= T1溢出率/2

若定时器/计数器T1工作在定时器模式0(16位自动重装载模式)，

如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH1, RL_TL1])/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH1, RL_TL1])/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

若定时器/计数器T1工作在模式2(8位自动重装模式)，

如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (256-TH1)/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk)/12/(256-TH1)/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1) / 2

0，不允许P3.4/T0管脚被配置为定时器1的时钟输出

B2 - T2CLKO：是否允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

1：允许将P3.0脚配置为定时器2的时钟输出T2CLKO，输出时钟频率=T2溢出率/2

如果T2_C/$\overline{T}$=0，定时器/计数器T2是对内部系统时钟计数，则：

T2工作在1T模式(AUXR.2/T2x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH2, RL_TL2])/2

T2工作在12T模式(AUXR.2/T2x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH2, RL_TL2])/2

如果T2_C/$\overline{T}$=1，定时器/计数器T2是对外部脉冲输入(P3.1/T2)计数，则：

输出时钟频率 = (T2_Pin_CLK) / (65536-[RL_TH2, RL_TL2])/2

0： 不允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

B4 - EX2：允许外部中断2(INT2)

B5 - EX3：允许外部中断3(INT3)

B6 - EX4：允许外部中断4(INT4)

3、辅助特殊功能寄存器：AUXR(地址：0x8E)

AUXR : Auxiliary register(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

B7 - T0x12：定时器0速度控制位。

0：定时器0速度是传统8051单片机定时器的速度，即12分频；

1：定时器0速度是传统8051单片机定时器速度的12倍，即不分频。

B6 - T1x12：定时器1速度控制位。

0：定时器1速度是传统8051单片机定时器的速度，即12分频；

1：定时器1速度是传统8051单片机定时器速度的12倍，即不分频。

如果串口1用T1作为波特率发生器，则由T1x12位决定串口1是12T还是1T。

B5 - UART_M0x6：串口1模式0的通信速度设置位。

0：串口1模式0的速度是传统8051单片机串口的速度，即12分频；

1：串口1模式0的速度是传统8051单片机串口速度的6倍，即2分频。

B4 - T2R：定时器2运行控制位。

0：不允许定时器2运行；

1：允许定时器2运行。

B3 - T2_C/T：控制定时器2用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T2/P3.1的外部脉冲进行计数)

B2 - T2x12：定时器2速度控制位

0，定时器2是传统8051单片机的速度，12分频；

1，定时器2的速度是传统8051单片机速度的12倍，不分频

如果串口1或串口2用T2作为波特率发生器，则由T2x12决定串口1或串口2是12T还是1T.

B1 - EXTRAM：内部/外部RAM存取控制位。

0：允许使用逻辑上在片外、物理上在片内的扩展RAM；

1：禁止使用逻辑上在片外、物理上在片内的扩展RAM。

B0 - S1ST2：串口1(UART1)选择定时器2作波特率发生器的控制位。

0：选择定时器1作为串口1(UART1)的波特率发生器；

1：选择定时器2作为串口1(UART1)的波特率发生器，此时定时器1得到释放，可以作为独立定时器使用。

4、定时器T4和T3的控制寄存器：T4T3M(地址：0xD1)

T4T3M(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO

B7 - T4R：定时器4运行控制位。
　　0：不允许定时器4运行；
　　1：允许定时器4运行。

B6 - T4_C/$\overline{T}$：控制定时器4用作定时器或计数器。
　　0，用作定时器(对内部系统时钟进行计数)；
　　1，用作计数器(对引脚T4/P0.7的外部脉冲进行计数)

B5 - T4x12：定时器4速度控制位。
　　0：定时器4速度是8051单片机定时器的速度，即12分频；
　　1：定时器4速度是8051单片机定时器速度的12倍，即不分频。

B4 - T4CLKO：是否允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO
　1：允许将P0.6脚配置为定时器4的时钟输出T4CLKO，输出时钟频率=T4溢出率/2
　　如果T4_C/$\overline{T}$=0，定时器/计数器T4是对内部系统时钟计数，则：
　　T4工作在1T模式(T4T3M.5/T4x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH4, RL_TL4])/2
　　T4工作在12T模式(T4T3M.5/T4x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH4, RL_TL4])/2
　　如果T4_C/$\overline{T}$=1，定时器/计数器T4是对外部脉冲输入(P0.7/T4)计数，则：
　　输出时钟频率 = (T4_Pin_CLK) / (65536-[RL_TH4, RL_TL4])/2
　0：不允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

B3 - T3R：定时器3运行控制位。
　　0：不允许定时器3运行；
　　1：允许定时器3运行。

B2 - T3_C/$\overline{T}$：控制定时器3用作定时器或计数器。
　　0，用作定时器(对内部系统时钟进行计数)；
　　1，用作计数器(对引脚T3/P0.5的外部脉冲进行计数)

B1 - T3x12：定时器3速度控制位。
　　0：定时器3速度是8051单片机定时器的速度，即12分频；
　　1：定时器3速度是8051单片机定时器速度的12倍，即不分频。

B0 - T3CLKO：是否允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO
　1：允许将P0.4脚配置为定时器3的时钟输出T3CLKO，输出时钟频率=T3溢出率/2
　　如果T3_C/$\overline{T}$=0，定时器/计数器T3是对内部系统时钟计数，则：
　　T3工作在1T模式(T4T3M.1/T3x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH3, RL_TL3])/2
　　T3工作在12T模式(T4T3M.1/T3x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH3, RL_TL3])/2
　　如果T3_C/$\overline{T}$=1，定时器/计数器T3是对外部脉冲输入(P0.5/T3)计数，则：
　　输出时钟频率 = (T3_Pin_CLK) / (65536-[RL_TH3, RL_TL3])/2
　0：不允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

STC MCU Limited

7.8.2 主时钟输出及其测试程序(C和汇编)

主时钟可以是内部高精度R/C时钟，也可以是外部输入的时钟或外部晶体振荡产生的时钟。由于单片机I/O口的对外输出速度最快不超过13.5MHz，所以可编程时钟对外输出速度最快也不超过13.5MHz，如果频率过高，需进行分频输出。

CLK_DIV (PCON2)：时钟分频寄存器(不可位寻址)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

如何利用MCLKO/P5.4或MCLKO_2/XTAL2/P1.6管脚输出时钟

MCLKO/P5.4或MCLKO_2/XTAL2/P1.6的时钟输出控制由CLK_DIV寄存器的MCKO_S1和MCKO_S0位控制。通过设置MCKO_S1(CLK_DIV.7)和MCKO_S0(CLK_DIV.6)可将MCLKO/P5.4管脚配置为主时钟输出同时还可以设置该主时钟的输出频率。

特殊功能寄存器：CLK_DIV (地址：97H)

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频， **输出时钟频率** = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频， **输出时钟频率** = MCLK / 4

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK是指主时钟频率。

STC15W4K60S4系列单片机在MCLKO/P5.4口或MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机均在MCLKO/P5.4口对外输出主时钟，且STC15W系列20-pin及其以上单片机除可在MCLKO/P5.4口对外输出主时钟外，还可在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

STC15W系列单片机通过CLK_DIV.3/MCLKO_2位来选择是在MCLKO/P5.4口对外输出主时钟，还是在MCLKO_2/XTAL2/P1.6口对外输出主时钟。

MCLKO_2：主时钟对外输出位置的选择位

0：在MCLKO/P5.4口对外输出主时钟；

1：在MCLKO_2/XTAL2/P1.6口对外输出主时钟；

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz，如果频率过高，需进行分频输出。

而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz，如果频率过高，需进行分频输出。

STC MCU Limited

下面是主时钟输出的示例程序：

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ----------------------------------------------------------------------*/
/* --- 演示STC15F2K60S2 系列单片机的主时钟输出 ----------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ---------------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ------------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;

#define   FOSC    18432000L
//-----------------------------------------
sfr       CLK_DIV       =       0x97;                 //时钟分频寄存器

//-----------------------------------------

void main()
{
          CLK_DIV       =       0x40;                 //0100,0000 P5.4输出频率为SYSclk
//        CLK_DIV       =       0x80;                 //1000,0000 P5.4输出频率为SYSclk/2
//        CLK_DIV       =       0xC0;                 //1100,0000 P5.4输出频率为SYSclk/4

          while (1);                                  //程序终止
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ----------------------------------------------------------------------*/
/* --- 演示STC15F2K60S2 系列单片机的主时钟输出 ----------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ---------------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ------------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

CLK_DIV         DATA    097H                    //IRC时钟输出控制寄存器

;-----------------------------------------
;interrupt vector table

        ORG     0000H
        LJMP    MAIN                            //复位入口
;-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,             #3FH            //initial SP
        MOV     CLK_DIV,        #40H            //0100,0000 P5.4输出频率为SYSclk
//      MOV     CLK_DIV,        #80H            //1000,0000 P5.4输出频率为SYSclk/2
//      MOV     CLK_DIV,        #C0H            //1100,0000 P5.4输出频率为SYSclk/4
        SJMP    $

//-----------------------------------------
        END
```

STC MCU Limited

7.8.3 定时器0对系统时钟或外部引脚T0的时钟输入进行可编程分频输出——及测试程序(C和汇编)

如何利用T0CLKO/P3.5管脚输出时钟

T0CLKO/P3.5管脚是否输出时钟由INT_CLKO (AUXR2)寄存器的T0CLKO位控制

AUXR2.0 - T0CLKO :　1, 允许时钟输出

0, 禁止时钟输出

T0CLKO的输出时钟频率由定时器0控制, 相应的定时器0需要工作在定时器的模式0(16位自动重装载模式)或模式2(8位自动重装载模式), 不要允许相应的定时器中断, 免得CPU反复进中断, 当然在特殊情况下也可允许相应的定时器中断。

新增加的特殊功能寄存器: INT_CLKO (AUXR2)(地址: 0x8F)

当T0CLKO/INT_CLKO.0=1时, P3.5/T1管脚配置为定时器0的时钟输出T0CLKO。

输出时钟频率 = T0 溢出率/2

若定时器/计数器T0工作在定时器模式0(16位自动重装载模式)时, (如下图所示)

如果C/$\overline{T}$=0, 定时器/计数器T0对内部系统时钟计数, 则:

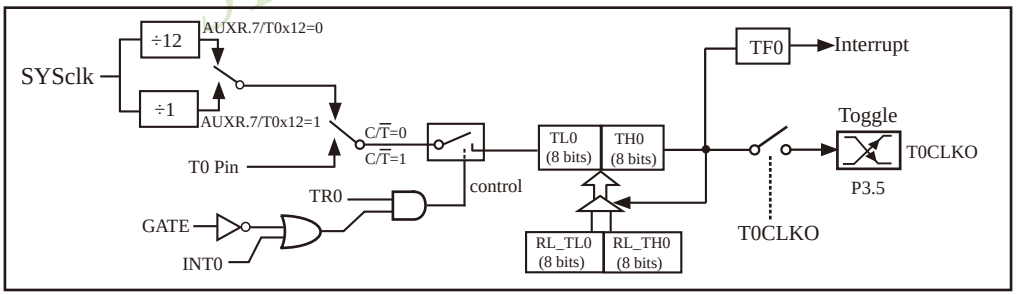
T0工作在1T模式(AUXR.7/T0x12=1)时的输出时钟频率 = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH0, RL_TL0])/2

如果C/$\overline{T}$=1, 定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数, 则:

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

RL_TH0为TH0的重装载寄存器, RL_TL0为TL0的重装载寄存器。



定时器/计数器0的模式 0: 16位自动重装

STC创新设计, 请不要再抄袭, 再抄袭就很无耻了

当T0CLKO/INT_CLKO.0=1且定时器/计数器T0工作在定时器模式2(8位自动重装模式)时, (如下图所示)

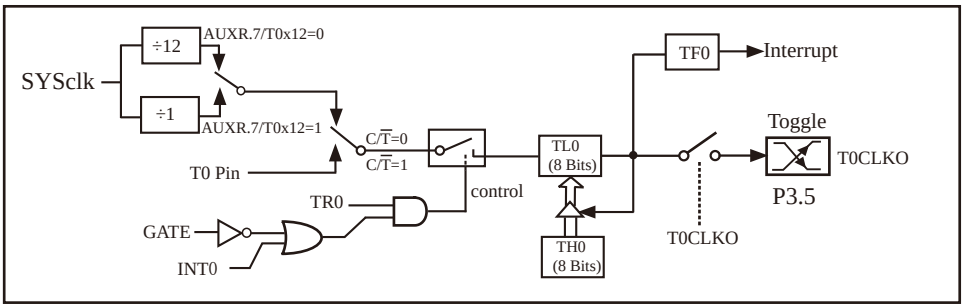
如果C/$\overline{T}$=0, 定时器/计数器T0对内部系统时钟计数, 则:

T0工作在1T模式(AUXR.7/T0x12=1)时的输出时钟频率=(SYSclk) / (256-TH0) / 2

T0工作在12T模式(AUXR.7/T0x12=0)时的输出时钟频率=(SYSclk) / 12 / (256-TH0) / 2

如果C/$\overline{T}$=1, 定时器/计数器T0是对外部脉冲输入(P3.4/T0)计数, 则:

输出时钟频率 = (T0_Pin_CLK) / (256-TH0) / 2



定时器/计数器0的模式 2: 8位自动重装

特别注意：对于STC15W1K16S系列和STC15W408S单片机，若要使用T0CLKO时钟输出功能，必须将P3.5口设置为强推挽输出模式。

下面是定时器0对内部系统时钟或外部引脚T0/P3.4的时钟输入进行可编程时钟分频输出的程序举例(C和汇编)：

1. C程序：

```
/*-----------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------------*/
/* --- 演示STC15F2K60S2 系列单片机定时器0的可编程时钟分频输出-----*/
/* --- 研发顾问QQ：800003751-------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ---------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*-----------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define FOSC 18432000L
//-----------------------------------------------
sfr     AUXR        =       0x8e;                   //辅助特殊功能寄存器
sfr     INT_CLKO    =       0x8f;                   //唤醒和时钟输出功能寄存器

sbit    T0CLKO      =       P3^5;                   //定时器0的时钟输出脚

#define F38_4KHz    (65536-FOSC/2/38400)            //1T模式
//#define F38_4KHz  (65536-FOSC/2/12/38400)         //12T模式
```

STC MCU Limited

```
//-----------------------------------------------
void main()
{
        AUXR    |=      0x80;                   //定时器0为1T模式
//      AUXR    &=      ~0x80;                  //定时器0为12T模式

        TMOD    =       0x00;                   //设置定时器为模式0(16位自动重装载)

        TMOD    &=      ~0x04;                  //C/T0=0, 对内部时钟进行时钟输出
//      TMOD    |=      0x04;                   //C/T0=1, 对T0引脚的外部时钟进行时钟输出

        TL0     =       F38_4KHz;               //初始化计时值
        TH0     =       F38_4KHz >> 8;
        TR0     =       1;
        INT_CLKO        =       0x01;           //使能定时器0的时钟输出功能

        while (1);                              //程序终止
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------*/
/* --- 演示STC15F2K60S2 系列单片机定时器0的可编程时钟分频输出-----*/
/* --- 研发顾问QQ：800003751-------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------*/
/* --- Tel: 86-755-82948412 --------------------------------------------------*/
/* --- Web: www.STCMCU.com ---------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*-------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR            DATA    08EH            //辅助特殊功能寄存器
INT_CLKO        DATA    08FH            //唤醒和时钟输出功能寄存器

T0CLKO          BIT     P3.5            //定时器0的时钟输出脚

F38_4KHz        EQU     0FF10H          //38.4KHz(1T模式下, 65536-18432000/2/38400)
//F38_4KHz      EQU     0FFECH          //38.4KHz(12T模式下, (65536-18432000/2/12/38400)
//-----------------------------------------------
```

STC MCU Limited

```
        ORG     0000H
        LJMP    MAIN                            //复位入口

//----------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #80H                    //定时器0为1T模式
//      ANL     AUXR,   #7FH                    //定时器0为12T模式

        MOV     TMOD,   #00H                    //设置定时器为模式0(16位自动重装载)

        ANL     TMOD,   #0FBH                   //C/T0=0, 对内部时钟进行时钟输出
//      ORL     TMOD,   #04H                    //C/T0=1, 对T0引脚的外部时钟进行时钟输出

        MOV     TL0,    #LOW F38_4KHz           //初始化计时值
        MOV     TH0,    #HIGH F38_4KHz
        SETB    TR0
        MOV     INT_CLKO,       #01H            //使能定时器0的时钟输出功能

        SJMP    $                               //程序终止

;----------------------------------------

        END
```

STC MCU Limited

7.8.4 定时器1对系统时钟或外部引脚T1的时钟输入进行可编程分频输出——及测试程序(C和汇编)

如何利用T1CLKO/P3.4管脚输出时钟

T1CLKO/P3.4管脚是否输出时钟由INT_CLKO (AUXR2)寄存器的T1CLKO位控制

AUXR2.1 - T1CLKO :　1, 允许时钟输出

0, 禁止时钟输出

T1CLKO的输出时钟频率由定时器1控制, 相应的定时器1需要工作在定时器的模式0(16位自动重装载模式)或模式2(8位自动重装载模式), 不要允许相应的定时器中断, 免得CPU反复进中断, 当然在特殊情况下也可允许相应的定时器中断。

新增加的特殊功能寄存器: INT_CLKO (AUXR2)(地址：0x8F)

当T1CLKO/INT_CLKO.1=1时，P3.4/T0管脚配置为定时器1的时钟输出T1CLKO。

输出时钟频率 = T1 溢出率/2

若定时器/计数器T1工作在定时器模式0(16位自动重装载模式)时，(如下图所示)

如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

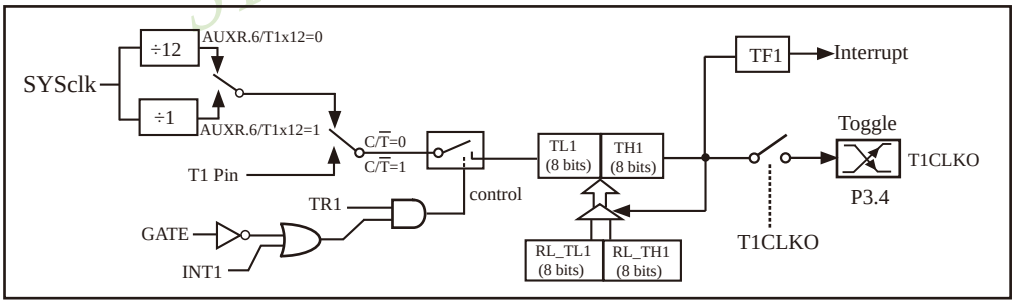
T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (65536-[RL_TH1, RL_TL1])/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk) /12/ (65536-[RL_TH1, RL_TL1])/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

RL_TH0为TH1的重装载寄存器，RL_TL1为TL0的重装载寄存器。



定时器/计数器1的模式 0: 16位自动重装

STC创新设计，请不要再抄袭，再抄袭就很无耻了

当T1CLKO/INT_CLKO.1=1且定时器/计数器T1工作在定时器模式2(8位自动重装模式)时，(如下图所示)

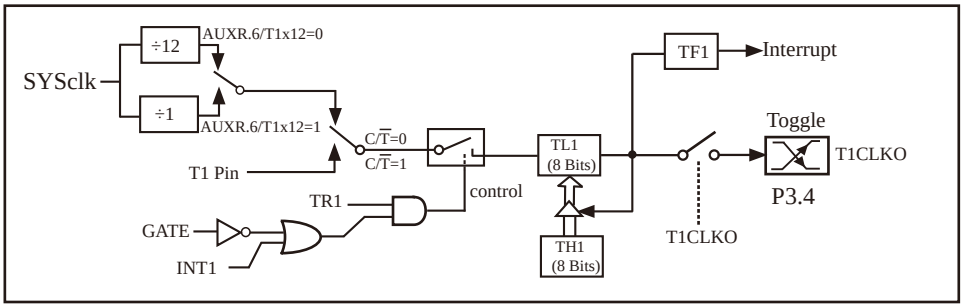
如果C/$\overline{T}$=0，定时器/计数器T1是对内部系统时钟计数，则：

T1工作在1T模式(AUXR.6/T1x12=1)时的输出频率 = (SYSclk) / (256-TH1)/2

T1工作在12T模式(AUXR.6/T1x12=0)时的输出频率 = (SYSclk)/12/(256-TH1)/2

如果C/$\overline{T}$=1，定时器/计数器T1是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1) / 2



定时器/计数器1的模式 2: 8位自动重装

下面是定时器1对内部系统时钟或外部引脚T1/P3.5的时钟输入进行可编程时钟分频输出的程序举例(C和汇编):

1. C程序:

```
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机定时器1的可编程时钟分频输出----------------------------------*/
/* --- 研发顾问QQ：800003751--------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 --------------------------------------------------------------*/
/* --- Web: www.STCMCU.com ---------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;

#define   FOSC    18432000L

//-----------------------------------------------
sfr   AUXR        =       0x8e;                   //辅助特殊功能寄存器
sfr   INT_CLKO    =       0x8f;                   //唤醒和时钟输出功能寄存器

sbit  T1CLKO      =       P3^4;                   //定时器1的时钟输出脚

#define   F38_4KHz        (65536-FOSC/2/38400)    //1T模式
//#define F38_4KHz        (65536-FOSC/2/12/38400) //12T模式
```

```
//-------------------------------------------
void main()
{
        AUXR    |=      0x40;                   //定时器1为1T模式
//      AUXR    &=      ~0x40;                  //定时器1为12T模式

        TMOD    =       0x00;                   //设置定时器为模式1(16位自动重装载)

        TMOD    &=      ~0x40;                  //C/T1=0, 对内部时钟进行时钟输出
//      TMOD    |=      0x40;                   //C/T1=1, 对T1引脚的外部时钟进行时钟输出

        TL1     =       F38_4KHz;               //初始化计时值
        TH1     =       F38_4KHz >> 8;
        TR1     =       1;
        INT_CLKO        =       0x02;           //使能定时器1的时钟输出功能

        while (1);                              //程序终止
}
```

2. 汇编程序：

```
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------------------------------------------------------------*/
/* --- 演示STC 15 系列单片机定时器1的可编程时钟分频输出----------------------*/
/* --- 研发顾问QQ：800003751--------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 --------------------------------------------------------------*/
/* --- Web: www.STCMCU.com ---------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-------*/
/*-------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR            DATA    08EH            //辅助特殊功能寄存器
INT_CLKO        DATA    08FH            //唤醒和时钟输出功能寄存器

T1CLKO          BIT     P3.4            //定时器1的时钟输出脚

F38_4KHz        EQU     0FF10H          //38.4KHz(1T模式下, 65536-18432000/2/38400)
//F38_4KHz      EQU     0FFECH          //38.4KHz(12T模式下, (65536-18432000/2/12/38400)
```

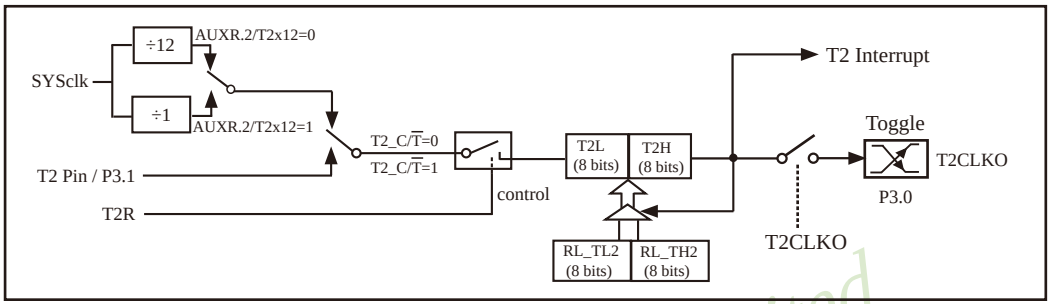
STC MCU Limited

```
        ORG     0000H
        LJMP    MAIN                        //复位入口

//-----------------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,  #40H                 //定时器1为1T模式
//      ANL     AUXR,  #0BFH                //定时器1为12T模式

        MOV     TMOD,  #00H                 //设置定时器为模式0(16位自动重装载)

        ANL     TMOD,  #0BFH                //C/T1=0, 对内部时钟进行时钟输出
//      ORL     TMOD,  #40H                 //C/T1=1, 对T1引脚的外部时钟进行时钟输出

        MOV     TL1,    #LOW F38_4KHz       //初始化计时值
        MOV     TH1,    #HIGH F38_4KHz
        SETB    TR1
        MOV     INT_CLKO,       #02H        //使能定时器1的时钟输出功能

        SJMP    $                           //程序终止

;-----------------------------------------------

        END
```

STC MCU Limited

7.8.5 定时器2对系统时钟或外部引脚T2的时钟输入进行可编程分频输出——及测试程序(C和汇编)

T2可以当定时器用，也可以当串口的波特率发生器和可编程时钟输出。
定时器2的原理框图如下：



定时器/计数器2的工作模式: 16位自动重装载

STC创新设计，请不要再抄袭，再抄袭就很无耻了

如何利用T2CLKO/P3.0管脚输出时钟

AUXR2.2 - T2CLKO：是否允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

1： 允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO，

0： 不允许将P3.0脚配置为定时器2(T2)的时钟输出T2CLKO

当T2CLKO/INT_CLKO.2=1时，P3.0管脚配置为定时器2的时钟输出T2CLKO。

输出时钟频率 = T2 溢出率/2

如果T2_C/$\overline{T}$=0，定时器/计数器T2对内部系统时钟计数，则：

T2工作在1T模式(AUXR.2/T2x12=1)时的输出时钟频率 = (SYSclk)/(65536-[RL_TH2, RL_TL2])/2

T2工作在12T模式(AUXR.2/T2x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH2, RL_TL2])/2

如果T2_C/$\overline{T}$=1，定时器/计数器T2是对外部脉冲输入(P3.1/T2)计数，则：

输出时钟频率 = (T2_Pin_CLK) / (65536-[RL_TH2, RL_TL2])/2

RL_TH2为T2H的重装载寄存器，RL_TL2为T2L的重装载寄存器。

用户在程序中如何具体设置T2CLKO/P3.0管脚输出时钟

1. 对定时器2寄存器T2H/T2L送16位重装载值, [T2H,T2L] = #reload_data
2. 对AUXR寄存器中的T2R位置1, 让定时器2运行
3. 对AUXR2/INT_CLKO寄存器中的T2CLKO位置1, 让定时器2的溢出在P3.0口输出时钟。

注意：当定时器/计数器2用作可编程时钟输出时，不要允许相应的定时器中断, 免得CPU反复进中断，在特殊情况下也可允许定时器/计数器2中断。

下面是定时器2对内部系统时钟或外部引脚T2/P3.1的时钟输入进行可编程时钟分频输出的程序举例(C和汇编):

1. C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2  系列  定时器2的可编程时钟分频输出举例----------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可---------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

typedef unsigned char   BYTE;
typedef unsigned int    WORD;

#define FOSC 18432000L

//-----------------------------------------------

sfr     AUXR        =   0x8e;               //辅助特殊功能寄存器
sfr     INT_CLKO    =   0x8f;               //唤醒和时钟输出功能寄存器
sfr     T2H         =   0xD6;               //定时器2高8位
sfr     T2L         =   0xD7;               //定时器2低8位

sbit    T2CLKO      =   P3^0;               //定时器2的时钟输出脚

#define F38_4KHz    (65536-FOSC/2/38400)    //1T模式
//#define F38_4KHz  (65536-FOSC/2/12/38400) //12T模式

//-----------------------------------------------

void main()
{
        AUXR    |=  0x04;                   //定时器2为1T模式
//      AUXR    &=  ~0x04;                  //定时器2为12T模式
```

```
    AUXR    &=      ~0x08;          //T2_C/T=0, 对内部时钟进行时钟输出
//  AUXR    |=      0x08;           //T2_C/T=1, 对T2(P3.1)引脚的外部时钟进行时钟输出

    T2L     =       F38_4KHz;                   //初始化计时值
    T2H     =       F38_4KHz >> 8;

    AUXR    |=      0x10;                       //定时器2开始计时
    INT_CLKO        =       0x04;               //使能定时器2的时钟输出功能

    while (1);                                  //程序终止
}
```

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2  系列  定时器2可编程时钟分频输出举例-------------------------*/
/* --- 研发顾问QQ：  800003751----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码, 请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码, 请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可---------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

AUXR            DATA    08EH            //辅助特殊功能寄存器
INT_CLKO        DATA    08FH            //唤醒和时钟输出功能寄存器
T2H             DATA    0D6H            //定时器2高8位
T2L             DATA    0D7H            //定时器2低8位

T2CLKO          BIT     P3.0            //定时器2的时钟输出脚

F38_4KHz        EQU     0FF10H          //38.4KHz(1T模式下，  65536-18432000/2/38400)
//F38_4KHz      EQU     0FFECH          //38.4KHz(12T模式下, (65536-18432000/2/12/38400)

//-----------------------------------------------
```

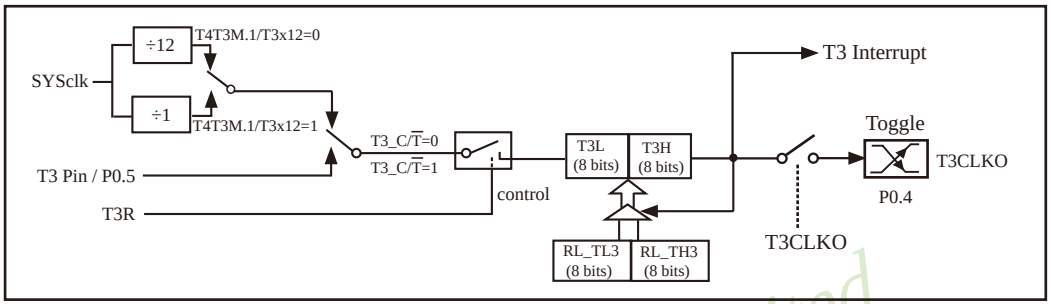
STC MCU Limited

```
        ORG     0000H
        LJMP    MAIN                                    //复位入口

//-----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #3FH

        ORL     AUXR,   #04H                            //定时器2为1T模式
//      ANL     AUXR,   #0FBH                           //定时器2为12T模式

        ANL     AUXR,   #0F7H           //T2_C/T=0, 对内部时钟进行时钟输出
//      ORL     AUXR,   #08H            //T2_C/T=1, 对T2(P3.1)引脚的外部时钟进行时钟输出

        MOV     T2L,    #LOW F38_4KHz                   //初始化计时值
        MOV     T2H,    #HIGH F38_4KHz
        ORL     AUXR,   #10H                            //定时器2开始计时
        MOV     INT_CLKO,   #04H                        //使能定时器2的时钟输出功能

        SJMP    $                                       //程序终止

;-----------------------------------------

        END
```

STC MCU Limited

7.8.6 定时器3对系统时钟或外部引脚T3的时钟输入进行可编程分频输出——及测试程序(C和汇编)

T3可以当定时器用，也可以当串口3的波特率发生器和可编程时钟输出。
定时器3的原理框图如下：



定时器/计数器3的工作模式: 16位自动重装载

STC创新设计，请不要再抄袭，再抄袭就很无耻了

如何利用T3CLKO/P0.4管脚输出时钟

T4T3M.0 - T3CLKO：是否允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

1： 允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO，

0： 不允许将P0.4脚配置为定时器3(T3)的时钟输出T3CLKO

当T3CLKO/T4T3M.0=1时，P0.4管脚配置为定时器3的时钟输出T3CLKO。

输出时钟频率 = T3 溢出率/2

如果T3_C/$\overline{T}$=0，定时器/计数器T3对内部系统时钟计数，则：

T3工作在1T模式(T4T3M.1/T3x12=1)时的输出时钟频率 = (SYSclk)/(65536-[RL_TH3, RL_TL3])/2

T3工作在12T模式(T4T3M.1/T3x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH3, RL_TL3])/2

如果T3_C/$\overline{T}$=1，定时器/计数器T3是对外部脉冲输入(P0.5/T3)计数，则：

输出时钟频率 = (T3_Pin_CLK) / (65536-[RL_TH3, RL_TL3])/2

RL_TH3为T3H的重装载寄存器，RL_TL3为T3L的重装载寄存器。

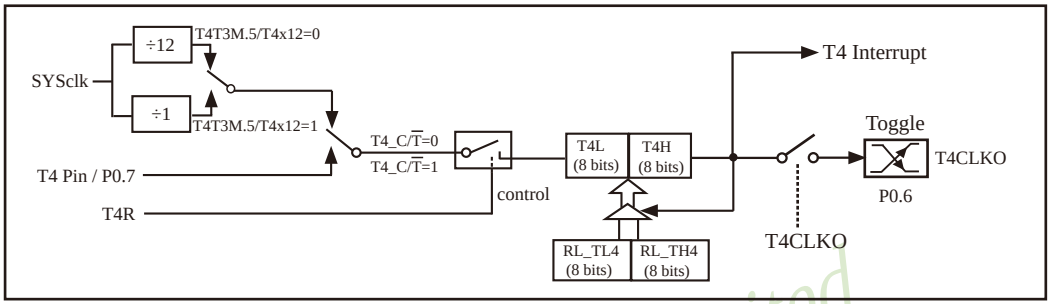
用户在程序中如何具体设置T3CLKO/P0.4管脚输出时钟

1. 对定时器3寄存器T3H/T3L送16位重装载值, [T3H,T3L] = #reload_data
2. 对T4T3M寄存器中的T3R位置1, 让定时器3运行
3. 对T4T3M寄存器中的T3CLKO位置1, 让定时器3的溢出在P0.4口输出时钟。

注意：当定时器/计数器3用作可编程时钟输出时，不要允许相应的定时器中断, 免得CPU反复进中断，在特殊情况下也可允许定时器/计数器3中断。

7.8.7 定时器4对系统时钟或外部引脚T4的时钟输入进行可编程分频输出——及测试程序(C和汇编)

T4可以当定时器用，也可以当串口4的波特率发生器和可编程时钟输出。
定时器4的原理框图如下：



定时器/计数器4的工作模式: 16位自动重装载

STC创新设计，请不要再抄袭，再抄袭就很无耻了

如何利用T4CLKO/P0.6管脚输出时钟

T4T3M.4 - T4CLKO：是否允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

1： 允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO，

0： 不允许将P0.6脚配置为定时器4(T4)的时钟输出T4CLKO

当T4CLKO/T4T3M.4=1时，P0.6管脚配置为定时器4的时钟输出T4CLKO。

输出时钟频率 = T4 溢出率/2

如果T4_C/$\overline{T}$=0，定时器/计数器T4对内部系统时钟计数，则：

T4工作在1T模式(T4T3M.5/T4x12=1)时的输出时钟频率 = (SYSclk)/(65536-[RL_TH4, RL_TL4])/2

T4工作在12T模式(T4T3M.5/T4x12=0)时的输出时钟频率=(SYSclk)/12/(65536-[RL_TH4, RL_TL4])/2

如果T4_C/$\overline{T}$=1，定时器/计数器T4是对外部脉冲输入(P0.7/T4)计数，则：

输出时钟频率 = (T4_Pin_CLK) / (65536-[RL_TH4, RL_TL4])/2

RL_TH4为T4H的重装载寄存器，RL_TL4为T4L的重装载寄存器。

用户在程序中如何具体设置T4CLKO/P0.6管脚输出时钟

1. 对定时器4寄存器T4H/T4L送16位重装载值, [T4H,T4L] = #reload_data
2. 对T4T3M寄存器中的T4R位置1, 让定时器4运行
3. 对T4T3M寄存器中的T4CLKO位置1, 让定时器4的溢出在P0.6口输出时钟。

注意：当定时器/计数器4用作可编程时钟输出时，不要允许相应的定时器中断, 免得CPU反复进中断，在特殊情况下也可允许定时器/计数器4中断。

7.8 掉电唤醒专用定时器及测试程序(C和汇编)

——进入掉电模式后可将单片机唤醒

——以15L开头的单片机进入掉电模式前必须启动掉电唤醒定时器

特别声明：以15L开头的芯片如需进入"掉电模式",进入"掉电模式"前必须启动掉电唤醒定时器<3uA>,不超过1秒要唤醒一次，以15F和15W开头的芯片以及新供货的STC15L2K60S2系列D版本芯片则不需要

STC15系列部分单片机新增了内部掉电唤醒定时器，在进入停机模式/掉电模式后，除了可以通过外部中断源进行唤醒外，还可以在无外部中断源的情况下通过使能内部掉电唤醒定时器定期唤醒CPU，使其恢复到正常工作状态。

掉电唤醒专用定时器的功耗：3V器件典型值低于3uA；5器件典型值低于5uA。

STC15系列单片机的内部低功耗掉电唤醒专用定时器由特殊功能寄存器WKTCH和WKTCL进行管理和控制。

WKTCL(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCL	AAH	name									1111 11110B

WKTCH(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCH	ABH	name	WKTEN								0111 1111B

内部掉电唤醒定时器是一个15位定时器，{WKTCH[6:0],WKTCL[7:0]}构成最长15位计数值(32768个),定时从0开始计数。

WKTEN：内部停机唤醒定时器的使能控制位。

WKTEN=1，允许内部停机唤醒定时器;

WKTEN=0，禁止内部停机唤醒定时器;

STC15系列有内部低功耗掉电唤醒专用定时器的单片机除增加了特殊功能寄存器WKTCL和WKTCH，还设计了了2个隐藏的特殊功能寄存器WKTCL_CNT和WKTCH_CNT来控制内部掉电唤醒专用定时器。WKTCL_CNT与WKTCL共用同一个地址，WKTCH_CNT与WKTCH共用同一个地址，WKTCL_CNT和WKTCH_CNT是隐藏的，对用户不可见。WKTCL_CNT和WKTCH_CNT实际上是作计数器使用，而WKTCL和WKTCH实际上作比较器使用。当用户对WKTCL和WKTCH写入内容时，该内容只写入寄存器WKTCL和WKTCH中，而不会写入WKTCL_CNT和WKTCH_CNT中。当用户读寄存器WKTCL和WKTCH中的内容时，实际上读的是寄存器WKTCL_CNT和WKTCH_CNT中的内容，而不是WKTCL和WKTCH中的内容。

STC MCU Limited

特殊功能寄存器WKTCL_CNT和WKTCH_CNT的格式如下所示：

WKTCL_CNT

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCL_CNT	AAH	name									1111 1111B

WKTCH_CNT

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCH_CNT	ABH	name	-								x111 1111B

通过软件将WKTCH寄存器中的WKTEN(Power Down Wakeup Timer Enable)位置‘1’,使能内部掉电唤醒专用定时器。一旦MCU进入Power Down Mode,内部掉电唤醒专用定时器[WKTCH_CNT,WKTCL_CNT]就从7FFFH开始计数,直到计数到与{WKTCH[6:0],WKTCL[7:0]}寄存器所设定的计数值相等后就让系统时钟开始振荡。如果主时钟使用的是内部系统时钟(由用户在ISP烧录程序时自行设置)，MCU在等待64个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU工作。如果主时钟使用的是外部晶体或时钟(由用户在ISP烧录程序时自行设置)，MCU在等待1024个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,才将时钟供给CPU工作。CPU获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。掉电唤醒之后，WKTCH_CNT和WKTCL_CNT的内容保持不变，因此可以通过读[WKTCH,WKTCL]的内容（实际上是读[WKTCH_CNT,WKTCL_CNT]的内容）读出单片机在停机模式/掉电模式所等待的时间。

这里请注意：用户在设置寄存器{WKTCH[6:0],WKTCL[7:0]}的计数值时，要按照所需要的计数次数，在计数次数的基础上减1所得的数值才是{WKTCH,WKTCL}的计数值。如用户需计数10次，则将9写入寄存器{WKTCH[6:0],WKTCL[7:0]}中。同样，如果用户需计数32768次，则应对{WKTCH[6:0],WKTCL[7:0]}写入7FFFH(即32767)。

内部掉电唤醒定时器有自己的内部时钟，其中掉电唤醒定时器计数一次的时间就是由该时钟决定的。内部掉电唤醒定时器的时钟频率约为32768Hz，当然误差较大。对于16-pin及其以上的单片机，用户可以通过读RAM区F8单元和F9单元的内容来获取内部掉电唤醒专用定时器常温下的时钟频率。对于8-pin单片机即STC15F101W系列，用户可以通过读RAM区78单元和79单元的内容来获取内部掉电唤醒专用定时器常温下的时钟频率。下面以16-pin及其以上的单片机为例，介绍如何计算内部掉电唤醒专用定时器的计数时间。

假设我们用[WIRC_H,WIRC_L]来表示从RAM区F8单元和F9单元获取到的内部掉电唤醒专用定时器常温下的时钟频率，则内部掉电唤醒专用定时器计数时间按下式计算：

$$\text{内部掉电唤醒专用定时器计数时间} = \frac{10^6\,\text{uS}}{[\text{WIRC\_H, WIRC\_L}]} \times 16 \times \text{计数次数}$$

例如：假设读到RAM区F8单元的内容为80H，F9单元的内容为00H，即内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为32768Hz，则内部掉电唤醒专用定时器最短计数时间(即计数一次的时间)为：$\frac{10^6\,uS}{32768}$ x 16 x 1 ≈ 488.28 uS

内部掉电唤醒专用定时器最长计数时间约为 488.28us x 32768 = 16S

设定{WKTCH[6:0],WKTCL[7:0]}寄存器的值等于9(即计数10次)且内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为32768Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 488.28uS x 10 ≈ 4882.8uS

设定{WKTCH[6:0],WKTCL[7:0]}寄存器的值等于32767(即最大计数值 = 32768 = 2^{15})且内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为32768Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 488.28uS x 32768 = 16S

下面给出了在读到RAM区F8单元的内容为80H，F9单元的内容为00H，即内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为32768Hz情况下，内部掉电唤醒专用定时器的计数时间：

{WKTCH[6:0],WKTCL[7:0]} = 0，	488.28uS x 1	= 488.28uS
{WKTCH[6:0],WKTCL[7:0]} = 9，	488.28uS x 10	= 4.8828mS
{WKTCH[6:0],WKTCL[7:0]} = 99，	488.28uS x 100	= 48.828mS
{WKTCH[6:0],WKTCL[7:0]} = 999，	488.28uS x 1000	= 488.28mS
{WKTCH[6:0],WKTCL[7:0]} = 4095，	488.28uS x 4096	= 2.0S
{WKTCH[6:0],WKTCL[7:0]} = 32767，	488.28uS x 32768=16S	

再假设读到RAM区F8单元的内容为79H，F9单元的内容为18H，即内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为31000Hz，则内部掉电唤醒专用定时器最短计数时间(即计数一次的时间)为：$\frac{10^6\,uS}{31000}$ x 16 x 1 ≈ 516.13 uS

内部掉电唤醒专用定时器最长计数时间约为 516.13us x 32768 ≈ 16.9S

设定{WKTCH[6:0],WKTCL[7:0]}寄存器的值等于9(即计数10次)且内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为31000Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 516.13uS x 10 ≈ 5161.3uS

下面给出了在读到RAM区F8单元的内容为79H，F9单元的内容为18H，即内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为31000Hz情况下，内部掉电唤醒专用定时器的计数时间：

{WKTCH[6:0],WKTCL[7:0]} = 0，	516.13uS x 1	≈ 516.13uS
{WKTCH[6:0],WKTCL[7:0]} = 9，	516.13uS x 10	≈ 5.1613mS
{WKTCH[6:0],WKTCL[7:0]} = 99，	516.13uS x 100	≈ 51.613mS

STC MCU Limited

{WKTCH[6:0],WKTCL[7:0]} = 999，　516.13uS x 1000 ≈ 516.13mS
{WKTCH[6:0],WKTCL[7:0]} = 4095，　516.13uS x 4096 ≈ 2.1S
{WKTCH[6:0],WKTCL[7:0]} = 32767，　516.13uS x 32768≈16.9S

又假设读到RAM区F8单元的内容为80H，F9单元的内容为E8H，即内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为33000Hz，则内部掉电唤醒专用定时器最短计数时间(即计数一次的时间)为：　$\frac{10^6\,uS}{33000}$ x 16 x 1 ≈ 484.85uS

内部掉电唤醒专用定时器最长计数时间约为 484.85us x 32768 ≈ 15.89S

设定{WKTCH[6:0],WKTCL[7:0]}寄存器的值等于9(即计数10次)且内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为33000Hz，则从系统掉电到启动系统振荡器，所需要等待的时间为 484.85uS x 10 ≈ 4848.5uS

下面给出了在读到RAM区F8单元的内容为80H，F9单元的内容为E8H，即内部掉电唤醒定时器的时钟频率[WIRC_H,WIRC_L]为33000Hz情况下，内部掉电唤醒专用定时器的计数时间：

{WKTCH[6:0],WKTCL[7:0]} = 0，　484.85uS x 1　≈ 484.85uS
{WKTCH[6:0],WKTCL[7:0]} = 9，　484.85uS x 10　≈ 4.8485mS
{WKTCH[6:0],WKTCL[7:0]} = 99，　484.85uS x 100　≈ 48.485mS
{WKTCH[6:0],WKTCL[7:0]} = 999，　484.85uS x 1000≈ 484.85mS
{WKTCH[6:0],WKTCL[7:0]} = 4095，　484.85uS x 4096≈ 1.986S
{WKTCH[6:0],WKTCL[7:0]} = 32767，　484.85uS x 32768≈15.89S

如果掉电唤醒定时器被允许（WKTEN=1），同时用户也将外部中断打开了。进入掉地模式后，当外部中断提前将单片机从停机模式唤醒时，可以通过读WKTCL和WKTCH的内容(实际是读WKTCL_CNT和WKTCH_CNT中的内容)，可以读出单片机在停机模式/掉电模式等待的时间。

STC MCU Limited

/*利用内部专用掉电唤醒定时器来唤醒掉电模式的示例程序（C程序）

1. C程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/-/* --- 演示STC15F2K60S2 系列 掉电唤醒定时器举例----------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

//-----------------------------------------------

sfr     WKTCL  =        0xaa;           //掉电唤醒定时器计时低字节
sfr     WKTCH =         0xab;           //掉电唤醒定时器计时高字节

sbit    P10     =       P1^0;

//-----------------------------------------------

void main()
{
        WKTCL  =        49;             //设置唤醒周期为488us*(49+1) = 24.4ms
        WKTCH =         0x80;           //使能掉电唤醒定时器

        while (1)
        {
                PCON = 0x02;            //进入掉电模式
                _nop_();
                _nop_();
                P10 = !P10;             //掉电唤醒后,取反测试口
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------------*/
/-/* --- 演示STC15F2K60S2 系列 掉电唤醒定时器举例----------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

WKTCL     DATA   0AAH                    //掉电唤醒定时器计时低字节
WKTCH     DATA   0ABH                    //掉电唤醒定时器计时高字节

//---------------------------------------

          ORG    0000H
          LJMP   MAIN                   //复位入口

//---------------------------------------

          ORG    0100H
MAIN:
          MOV    SP,     #3FH

          MOV    WKTCL, #49             //设置唤醒周期为488us*(49+1) = 24.4ms
          MOV    WKTCH,#80H             //使能掉电唤醒定时器
LOOP:
          MOV    PCON,   #02H           //进入掉电模式
          NOP
          NOP
          CPL    P1.0                   //掉电唤醒后,取反测试口
          JMP    LOOP

          SJMP   $

;---------------------------------------

          END
```

STC MCU Limited

7.9 外部管脚T0/T1/T2/T3/T4如何唤醒掉电模式/停机模式

如果定时器(T0/T1/T2/T3/T4)的中断在进入掉电模式/停机模式前已经被允许，即ET0/ET1/ET2/ET3/ET4及EA在进入掉电模式/停机模式前已经被置为1，则进入掉电模式/停机模式后定时器仍继续工作，且定时器T0/T1/T2/T3/T4的外部管脚(T0/P3.4, T1/P3.5, T2/P3.1, T3/P0.5, T4/P0.7)如发生由高到低的变化可以将MCU从掉电模式/停机模式唤醒。当MCU由定时器T0/T1/T2/T3/T4的外部管脚由高到低的变化唤醒时，如果主时钟使用的是内部系统时钟(由用户在ISP烧录程序时自行设置)，MCU在等待64个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU工作；如果主时钟使用的是外部晶体或时钟(由用户在ISP烧录程序时自行设置)，MCU在等待1024个时钟后，就认为此时系统时钟从开始起振的不稳定状态已经过渡到稳定状态,就将时钟供给CPU工作；CPU获得时钟后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行，不进入相应定时器的中断程序。

STC MCU Limited

第8章 串行口通信

除STC15F101W系列无串行口功能外，其他STC15系列单片机都有串行口功能，其中STC-15W4K60S4系列单片机有4个高速异步串行通信端口、STC15F2K60S2系列单片机有2个高速异步串行通信端口、STC15W1K16S/STC15W408S/STC15W408AS/STC15W201S/STC15F408AD系列单片机有1个高速异步串行通信端口，如下表所示：

下表总结了STC15系列单片机内置了高速异步串行通信端口的单片机型号：

高速异步串行通信端口 / 单片机型号	串行口1	串行口2	串行口3	串行口4
STC15W4K60S4系列	√	√	√	√
STC15F2K60S2系列	√	√		
STC15W1K16S系列	√			
STC15W408S系列	√			
STC15W408AS系列	√			
STC15W201S系列	√			
STC15F408AD系列	√			
STC15F101W系列				

上表中√表示对应的系列有相应的串行口。

STC MCU Limited

现以STC15W4K60S4系列单片机为例，介绍STC15系列单片机的串行通信端口。

STC15W4K60S4系列单片机具有4个采用UART(Universal Asychronous Receiver/Transmitter)工作方式的全双工异步串行通信接口(串口1、串口2、串口3和串口4)。每个串行口由2个数据缓冲器、一个移位寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由2个互相独立的接收、发送缓冲器构成，可以同时发送和接收数据。发送缓冲器只能写入而不能读出，接收缓冲器只能读出而不能写入，因而两个缓冲器可以共用一个地址码。串行口1的两个缓冲器共用的地址码是99H；串行口2的两个缓冲器共用的地址码是9BH；串行口3的两个缓冲器共用的地址码是ADH；串行口4的两个缓冲器共用的地址码是85H。串行口1的两个缓冲器统称串行通信特殊功能寄存器SBUF；串行口2的两个缓冲器统称串行通信特殊功能寄存器S2BUF；串行口3的两个缓冲器统称串行通信特殊功能寄存器S3BUF；串行口4的两个缓冲器统称串行通信特殊功能寄存器S4BUF。

STC15W4K60S4系列单片机的串行口1有4种工作方式，其中两种方式的波特率是可变的，另两种是固定的，以供不同应用场合选用。串行口2/串行口3/串行口4都只有两种工作方式，这两种方式的波特率都是可变的。用户可用软件设置不同的波特率和选择不同的工作方式。主机可通过查询或中断方式对接收/发送进行程序处理，使用十分灵活。

STC15W4K60S4系列单片机串行口1对应的硬件部分是TxD和RxD。串行口1可以在3组管

脚之间进行切换。通过设置特殊功能寄存器AUXR1/P_SW1中的位S1_S1/AUXR1.7和S1_S0/P_SW1.6，可以将串行口1从[RxD/P3.0,TxD/P3.1]切换到[RxD_2/P3.6,TxD_2/P3.7]，还可以切换到[RxD_3/P1.6/XTAL2,TxD_3/P1.7/XTAL1]。注意，当串行口1在[RxD_2/P1.6, TxD_2/P1.7]时，系统要使用内部时钟。串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1]上。

STC15W4K60S4系列单片机串行口2对应的硬件部分是TxD2和RxD2。串行口2可以在2组管脚之间进行切换。通过设置特殊功能寄存器P_SW2中的位S2_S/P_SW2.0，可以将串行口2从[RxD2/P1.0,TxD2/P1.1]切换到[RxD2_2/P4.6,TxD2_2/P4.7]。

STC15W4K60S4系列单片机串行口3对应的硬件部分是TxD3和RxD3。串行口3可以在2组管脚之间进行切换。通过设置特殊功能寄存器P_SW2中的位S3_S/P_SW2.1，可以将串行口3从[RxD3/P0.0,TxD3/P0.1]切换到[RxD3_2/P5.0,TxD3_2/P5.1]。

STC15W4K60S4系列单片机串行口4对应的硬件部分是TxD4和RxD4。串行口4可以在2组管脚之间进行切换。通过设置特殊功能寄存器P_SW2中的位S4_S/P_SW2.2，可以将串行口4从[RxD4/P0.2,TxD4/P0.3]切换到[RxD4_2/P5.2,TxD4_2/P5.3]。

STC15W4K60S4系列单片机的串行通信口，除用于数据通信外，还可方便地构成一个或多个并行I/O口，或作串—**并转换，或用于扩展串行外设等。**

STC MCU Limited

8.1 串行口1的相关寄存器

符号	描述	地址	位地址及符号 MSB							LSB	复位值
T2H	定时器2高8位寄存器	D6H									0000 0000B
T2L	定时器2低8位寄存器	D7H									0000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000 0001B
SCON	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
SBUF	Serial Buffer	99H									xxxx xxxxB
PCON	Power Control	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
SADEN	Slave Address Mask	B9H									0000 0000B
SADDR	Slave Address	A9H									0000 0000B
AUXR1 P_SW1	辅助寄存器1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000B
CLK_DIV PCON2	时钟分频寄存器	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B

STC MCU Limited

1. 串行口1的控制寄存器SCON和PCON

STC15系列单片机的串行口1设有两个控制寄存器：串行控制寄存器SCON和波特率选择特殊功能寄存器PCON。

串行控制寄存器SCON用于选择串行通信的工作方式和某些控制功能。其格式如下：

SCON：串行控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE：当PCON寄存器中的SMOD0/PCON.6位为1时，该位用于帧错误检测。当检测到一个无效停止位时，通过UART接收器设置该位。它必须由软件清零。

当PCON寄存器中的SMOD0/PCON.6位为0时，该位和SM1一起指定串行通信的工作方式，如下表所示。

其中SM0、SM1按下列组合确定串行口1的工作方式：

SM0	SM1	工作方式	功能说明	波特率
0	0	方式0	同步移位串行方式：移位寄存器	当UART_M0x6 = 0时，波特率是SYSclk/12, 当UART_M0x6 = 1时，波特率是SYSclk / 2
0	1	方式1	8位UART， 波特率可变	串行口1用定时器1作为其波特率发生器且定时器1工作于模式0(16位自动重装载模式)或串行口用定时器2作为其波特率发生器时，波特率=(定时器1的溢出率或定时器T2的溢出率)/4. 注意：此时波特率与SMOD无关。 当串行口1用定时器1作为其波特率发生器且定时器1工作于模式2(8位自动重装模式)时，波特率=(2^{SMOD}/32)×(定时器1的溢出率)
1	0	方式2	9位UART	(2^{SMOD} / 64) x SYSclk系统工作时钟频率
1	1	方式3	9位UART， 波特率可变	当串行口1用定时器1作为其波特率发生器且定时器1工作于模式0(16位自动重装载模式)或串行口用定时器2作为其波特率发生器时，波特率=(定时器1的溢出率或定时器T2的溢出率)/4. 注意：此时波特率与SMOD无关。 当串行口1用定时器1作为其波特率发生器且定时器1工作于模式2(8位自动重装模式)时，波特率=(2^{SMOD}/32)×(定时器1的溢出率)

当定时器1工作于模式0（16位自动重装载模式）且AUXR.6/T1x12 = 0时，
定时器1的溢出率 = SYSclk/12/(65536 - [RL_TH1,RL_TL1]) ;
当定时器1工作于模式0（16位自动重装载模式）且AUXR.6/T1x12 = 1时,
定时器1的溢出率 = SYSclk / (65536 - [RL_TH1,RL_TL1])

当定时器1工作于模式2（8位自动重装模式）且T1x12 = 0时，
定时器1的溢出率 = SYSclk/12/(256 - TH1);
当定时器1工作于模式2（8位自动重装模式）且T1x12 = 1时,
定时器1的溢出率 = SYSclk / (256 - TH1)

当AUXR.2/T2x12 = 0时, 定时器T2的溢出率 = SYSclk / 12/ (65536 - [RL_TH2,RL_TL2]);
当AUXR.2/T2x12 = 1时, 定时器T2的溢出率 = SYSclk / (65536 - [RL_TH2,RL_TL2]);

SM2：允许方式2或方式3多机通信控制位。
在方式2或方式3时，如果SM2位为1且REN位为1，则接收机处于地址帧筛选状态。此时可以利用接收到的第9位(即RB8)来筛选地址帧：若RB8=1，说明该帧是地址帧，地址信息可以进入SBUF，并使RI为1，进而在中断服务程序中再进行地址号比较；若RB8=0，说明该帧不是地址帧，应丢掉且保持RI=0。在方式2或方式3中，如果SM2位为0且REN位为1，接收收机处于地址帧筛选被禁止状态。不论收到的RB8为0或1，均可使接收到的信息进入SBUF,并使RI=1,此时RB8通常为校验位.
方式1和方式0是非多机通信方式，在这两种方式时，要设置SM2 应为0。

REN：允许/禁止串行接收控制位。由软件置位REN，即REN=1为允许串行接收状态，可启动串行接收器RxD，开始接收信息。软件复位REN，即REN=0，则禁止接收。

TB8：在方式2或方式3，它为要发送的第9位数据，按需要由软件置位或清0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式0和方式1中，该位不用.

RB8：在方式2或方式3，是接收到的第9位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式0中不用RB8(置SM2=0). 方式1中也不用RB8(置SM2=0, RB8是接收到的停止位)。

TI：发送中断请求标志位。在方式0，当串行发送数据第8位结束时，由内部硬件自动置位，即TI=1，向主机请求中断，响应中断后TI必须用软件清零，即TI=0。在其他方式中，则在停止位开始发送时由内部硬件置位，即TI=1, 响应中断后TI必须用软件清零。

RI：接收中断请求标志位。在方式0，当串行接收到第8位结束时由内部硬件自动置位RI=1，向主机请求中断，响应中断后RI必须用软件清零，即RI=0。在其他方式中，串行接收到停止位的中间时刻由内部硬件置位，即RI=1, 向CPU发中断申请，响应中断后RI必须由软件清零。

SCON的所有位可通过整机复位信号复位为全“0”。SCON的字节地址为98H，可位寻址，各位地址为98H~~9FH，可用软件实现位设置。

串行通信的中断请求：当一帧发送完成，内部硬件自动置位TI，即TI=1，请求中断处理；当接收完一帧信息时，内部硬件自动置位RI，即RI=1，请求中断处理。由于TI和RI以**“或逻辑”关系向主机请求中断，所以主机响应中断时事先并不知道是**TI还是RI请求的中断，必须在中断服务程序中查询TI和RI进行判别，然后分别处理。因此，两个中断请求标志位均不能由硬件自动置位，必须通过软件清0，否则将出现一次请求多次响应的错误。

STC MCU Limited

电源控制寄存器PCON中的SMOD/PCON. 7用于设置方式1、方式2、方式3的波特率是否加倍。

电源控制寄存器PCON格式如下：

PCON：电源控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

SMOD：波特率选择位。当用软件置位SMOD，即SMOD=1，则使串行通信方式1、2、3的波特率加倍；SMOD=0，则各工作方式的波特率不加倍。复位时SMOD=0。

SMOD0：帧错误检测有效控制位。当SMOD0=1，SCON寄存器中的SM0/FE位用于FE(帧错误检测)功能；当SMOD0=0，SCON寄存器中的SM0/FE位用于SM0功能, 和SM1一起指定串行口的工作方式。复位时SMOD0=0

PCON中的其他位都与串行口1无关，在此不作介绍。

2. 串行口数据缓冲寄存器SBUF

STC15系列单片机的串行口1缓冲寄存器(SBUF)的地址是99H，实际是2个缓冲器，写SBUF的操作完成待发送数据的加载，读SBUF的操作可获得已接收到的数据。两个操作分别对应两个不同的寄存器，1个是只写寄存器，1个是只读寄存器。

串行通道内设有数据寄存器。在所有的串行通信方式中，在写入SBUF信号(MOV　SBUF,A)的控制下，把数据装入相同的9位移位寄存器，前面8位为数据字节，其最低位为移位寄存器的输出位。根据不同的工作方式会自动将“1”或TB8的值装入移位寄存器的第9位,并进行发送.

串行通道的接收寄存器是一个输入移位寄存器。在方式0时它的字长为8位，其他方式时为9位。当一帧接收完毕，移位寄存器中的数据字节装入串行数据缓冲器SBUF中,其第9位则装入SCON寄存器中的RB8位。如果由于SM2使得已接收到的数据无效时,RB8和SBUF中内容不变.

由于接收通道内设有输入移位寄存器和SBUF缓冲器，从而能使一帧接收完将数据由移位寄存器装入SBUF后，可立即开始接收下一帧信息，主机应在该帧接收结束前从SBUF缓冲器中将数据取走，否则前一帧数据将丢失。SBUF以并行方式送往内部数据总线。

3. 辅助寄存器AUXR

辅助寄存器AUXR的格式及各位含义如下:

AUXR：辅助寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2

T0x12：定时器0速度控制位

0，定时器0是传统8051速度，12分频;

1，定时器0的速度是传统8051的12倍，不分频

T1x12：定时器1速度控制位

0，定时器1是传统8051速度，12分频;

1，定时器1的速度是传统8051的12倍，不分频

如果UART1/串口1用T1作为波特率发生器，则由T1x12决定UART1/串口是12T还是1T

UART_M0x6：串口模式0的通信速度设置位。

0，串口1模式0的速度是传统8051单片机串口的速度，12分频;

1，串口1模式0的速度是传统8051单片机串口速度的6倍，2分频

T2R：定时器2允许控制位

0，不允许定时器2运行;

1，允许定时器2运行

T2_C/$\overline{T}$：控制定时器2用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数);

1，用作计数器(对引脚T2/P3.1的外部脉冲进行计数)

STC MCU Limited

T2x12：定时器2速度控制位

0，定时器2是传统8051速度，12分频；

1，定时器2的速度是传统8051的12倍，不分频

如果串口1或串口2用T2作为波特率发生器，则由T2x12决定串口1或串口2是12T还是1T.

EXTRAM：内部/外部RAM存取控制位

0，允许使用逻辑上在片外、物理上在片内的扩展RAM；

1，禁止使用逻辑上在片外、物理上在片内的扩展RAM

S1ST2：串口1(UART1)选择定时器2作波特率发生器的控制位

0，选择定时器1作为串口1(UART1)的波特率发生器；

1，选择定时器2作为串口1(UART1)的波特率发生器，此时定时器1得到释放，可以作为独立定时器使用

串口1可以选择定时器1做波特率发生器，也可以选择**定时器2作为波特率发生器。**当设置AUXR寄存器中的S1ST2位(串行**口波特率选择位)为1时，串行口1选择定时器2作为波特率发生**器,此时定时器1可以释放出来作为定时器/计数器/时钟输出使用.

对于**STC15系列单片机，串口2只能使用定时器2作为波特率发生器，不能够选择其他定时**器作为其波特率发生器；而串口1默认选择定时器2作为其波特率发生器，也可以选择定时器1作为其波特率发生器；串口3默认选择定时器2作为其波特率发生器，也可以选择定时器3作为其波特率发生器；串口4默认选择定时器2作为其波特率发生器，也可以选择定时器4作为其波特率发生器。

4. 定时器2的寄存器T2H, T2L

定时器2寄存器T2H(地址为D6H，复位值为00H)及寄存器T2L(地址为D7H，复位值为00H)用于保存重装时间常数。

注意：对于STC15系列单片机，串口2只能使用定时器2作为波特率发生器，不能够选择其他定时器作为其波特率发生器；而串口1默认选择定时器2作为其波特率发生器，也可以选择定时器1作为其波特率发生器；串口3默认选择定时器2作为其波特率发生器，也可以选择定时器3作为其波特率发生器；串口4默认选择定时器2作为其波特率发生器，也可以选择定时器4作为其波特率发生器。

5. 从机地址控制寄存器SADEN和SADDR

为了方便多机通信，STC15系列单片机设置了从机地址控制寄存器SADEN和SADDR。其中SADEN是从机地址掩模寄存器(地址为B9H，复位值为00H)，SADDR是从机地址寄存器(地址为A9H，复位值为00H)。

6. 与串行口1中断相关的寄存器位ES和PS

串行口中断允许位ES位于中断允许寄存器IE中，中断允许寄存器的格式如下：

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的总中断允许控制位

EA=1，CPU开放中断，

EA=0，CPU屏蔽所有的中断申请。

EA的作用是使中断允许形成多级控制。即各中断源首先受EA控制;其次还受各中断源自己的中断允许控制位控制。

ES：串行口中断允许位

ES=1，允许串行口中断，

ES=0，禁止串行口中断。

IP：中断优先级控制寄存器低（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PS:　串行口1中断优先级控制位。

当PS=0时，串行口1中断为最低优先级中断(优先级0)

当PS=1时，串行口1中断为最高优先级中断(优先级1)

STC MCU Limited

7. 将串口1进行切换的寄存器AUXR1(P_SW1)

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,0000

串口1/S1可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD,P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2,P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1] 串口1在P1口时要使用内部时钟
1	1	无效

串口1建议放在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1]上。

CCP可在3个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP可在P1/P2/P3之间来回切换
0	0	CCP在[P1.2/ECI,P1.1/CCP0,P1.0/CCP1,P3.7/CCP2]
0	1	CCP在[P3.4/ECI_2,P3.5/CCP0_2,P3.6/CCP1_2,P3.7/CCP2_2]
1	0	CCP在[P2.4/ECI_3,P2.5/CCP0_3,P2.6/CCP1_3,P2.7/CCP2_3]
1	1	无效

SPI可在3个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI可在P1/P2/P4之间来回切换
0	0	SPI在[P1.2/SS,P1.3/MOSI,P1.4/MISO,P1.5/SCLK]
0	1	SPI在[P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI在[P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	无效

8. 串口1的中继广播方式设置位——Tx_Rx/CLK_DIV.4

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

串口1的RxD管脚和TxD管脚可以在3组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1];
[RxD_2/P3.6, TxD_2/P3.7];
[RxD_3/P1.6, TxD_3/P1.7].

STC MCU Limited

8.2 串行口1工作模式

STC15系列单片机的串行通信接口有4种工作模式，可通过软件编程对SCON中的SM0、SM1的设置进行选择。其中模式1、模式2和模式3为异步通信，每个发送和接收的字符都带有1个启动位和1个停止位。在模式0中，串行口被作为1个简单的移位寄存器使用。

8.2.1 串行口1工作模式0：同步移位寄存器(建议初学者不学)

在模式0状态，串行通信接口工作在同步移位寄存器模式，当串行口模式0的通信速度设置位UART_M0x6/AUXR.5 = 0时，其波特率固定为SYSclk/12。当串行口模式0的通信速度设置位UART_M0x6/AUXR.5 = 1时，其波特率固定为SYSclk/2。串行口数据由RxD/P3.0端输入，同步移位脉冲（SHIFTCLOCK）由TxD/P3.1输出，发送、接收的是8位数据，低位在先。

模式0的发送过程：当主机执行将数据写入发送缓冲器SBUF指令时启动发送，串行口即将8位数据以SYSclk/12或SYSclk/2(由UART_M0x6/AUXR.5确定是12分频还是2分频)的波特率从RxD管脚输出(从低位到高位)，发送完中断标志TI置″1″，TxD管脚输出同步移位脉冲（SHIFT-CLOCK）。波形如图8-1中“发送”所示。

当写信号有效后，相隔一个时钟，发送控制端SEND有效(高电平)，允许RxD发送数据，同时允许TxD输出同步移位脉冲。一帧(8位)数据发送完毕时，各控制端均恢复原状态，只有TI保持高电平，呈中断申请状态。在再次发送数据前，必须用软件将TI清0。

模式0接收过程：模式0接收时，复位接收中断请求标志RI，即RI＝0，置位允许接收控制位REN=1时启动串行模式0接收过程。启动接收过程后，RxD为串行输入端，TxD为同步脉冲输出端。串行接收的波特率为SYSclk/12或SYSclk/2(由UART_M0x6/AUXR.5确定是12分频还是2分频)。其时序图如图8-1中“接收”所示。

当接收完成一帧数据(8位)后，控制信号复位，中断标志RI被置″1″，呈中断申请状态。当再次接收时，必须通过软件将RI清0

工作于模式0时，必须清0多机通信控制位SM2，使不影响TB8位和RB8位。由于波特率固定为SYSclk/12或SYSclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串行口工作模式0的示意图如图8-1所示

由示意图中可见，由TX和RX控制单元分别产生中断请求信号并置位TI=1或RI =1，经**“或门”送主机请求中断，所以主机响应中断后必须软件判别是**TI还是RI请求中断，必须软件清0中断请求标志位TI或RI。

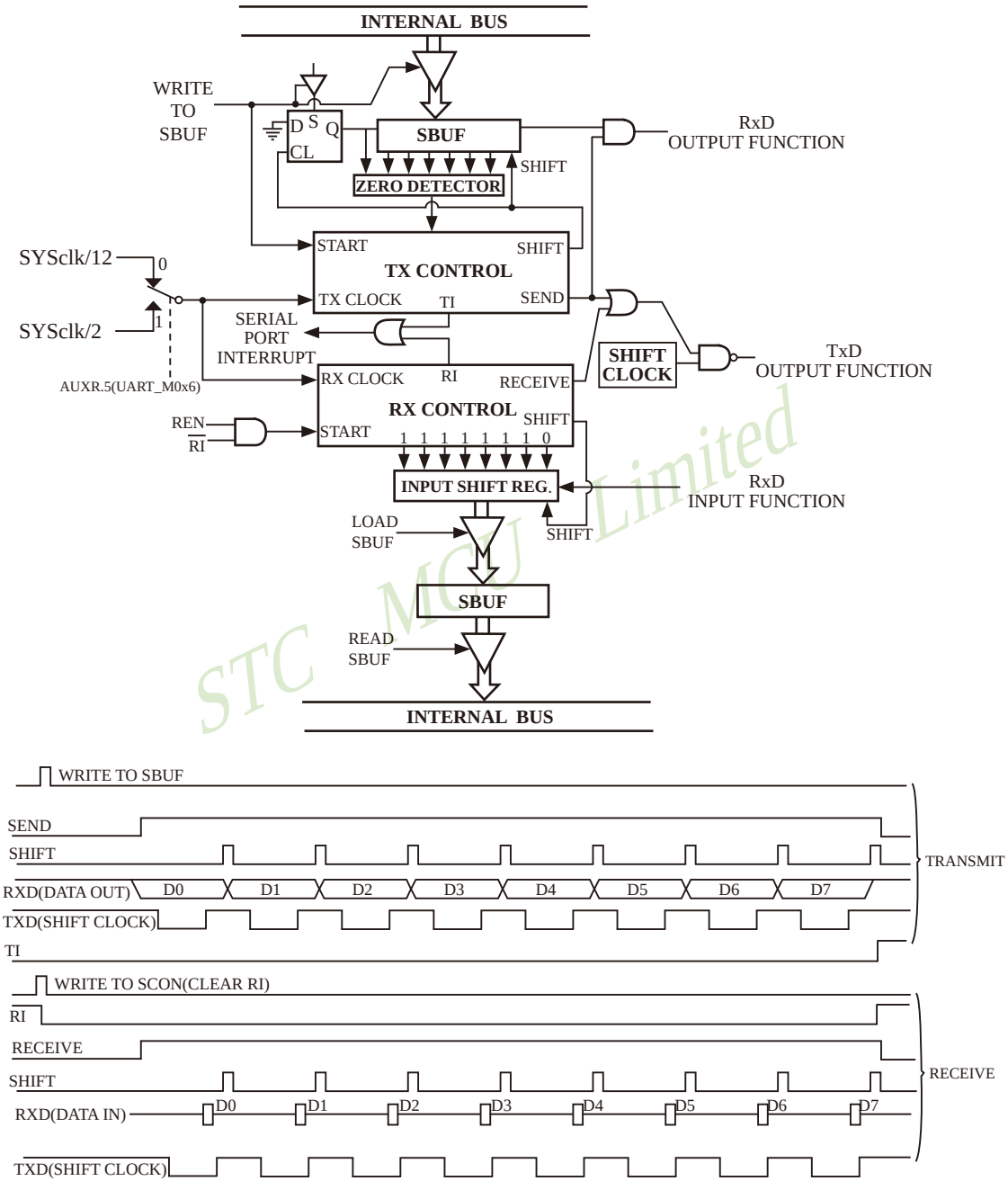


图8-1　串行口1模式0功能结构及时序示意图

8.2.2 串行口1工作模式1：8位UART，波特率可变

当软件设置SCON的SM0、SM1为“01”时，串行口1则以模式1工作。此模式为8位UART格式，一帧信息为10位：1位起始位，8位数据位(低位在先)和1位停止位。波特率可变，即可根据需要进行设置。TxD/P3.1为发送信息，RxD/P3.0为接收端接收信息，串行口为全双工接受/发送串行口。

图8-2为串行模式1的功能结构示意图及接收/发送时序图

模式1的发送过程：串行通信模式发送时，数据由串行发送端TxD输出。当主机执行一条写“SBUF“的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第9位，并通知TX控制单元开始发送。发送各位的定时是由16分频计数器同步。

移位寄存器将数据不断右移送TxD端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第9位“1”，在它的左边各位全为“0”，这个状态条件，使TX控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位TI，即TI=1，向主机请求中断处理。

模式1的接收过程：当软件置位接收允许标志位REN，即REN=1时，接收器便以选定波特率的16分频的速率采样串行接收端口RxD，当检测到RxD端口从“1”→“0”的负跳变时就启动接收器准备接收数据，并立即复位16分频计数器，将1FFH植装入移位寄存器。复位16分频计数器是使它与输入位时间同步。

16分频计数器的16个状态是将1波特率（每位接收时间）均为16等份，在每位时间的7、8、9状态由检测器对RxD端口进行采样，所接收的值是这次采样直经“三中取二”的值，即3次采样至少2次相同的值，以此消除干扰影响，提高可靠性。在起始位，如果接收到的值不为“0”（低电平），则起始位无效，复位接收电路，并重新检测″1″→″0″的跳变。如果接收到的起始位有效，则将它输入移位寄存器，并接收本帧的其余信息。

接收的数据从接收移位寄存器的右边移入，已装入的1FFH向左边移出，当起始位″0″移到移位寄存器的最左边时，使RX控制器作最后一次移位，完成一帧的接收。若同时满足以下两个条件：

- RI=0；
- SM2=0或接收到的停止位为1。

则接收到的数据有效，实现装载入SBUF，停止位进入RB8，置位RI，即RI=1，向主机请求中断，若上述两条件不能同时满足，则接收到的数据作废并丢失，无论条件满足与否，接收器重又检测RxD端口上的″1″→″0″的跳变，继续下一帧的接收。接收有效，在响应中断后，必须由软件清0，即RI=0。通常情况下，串行通信工作于模式1时，SM2设置为″0″。

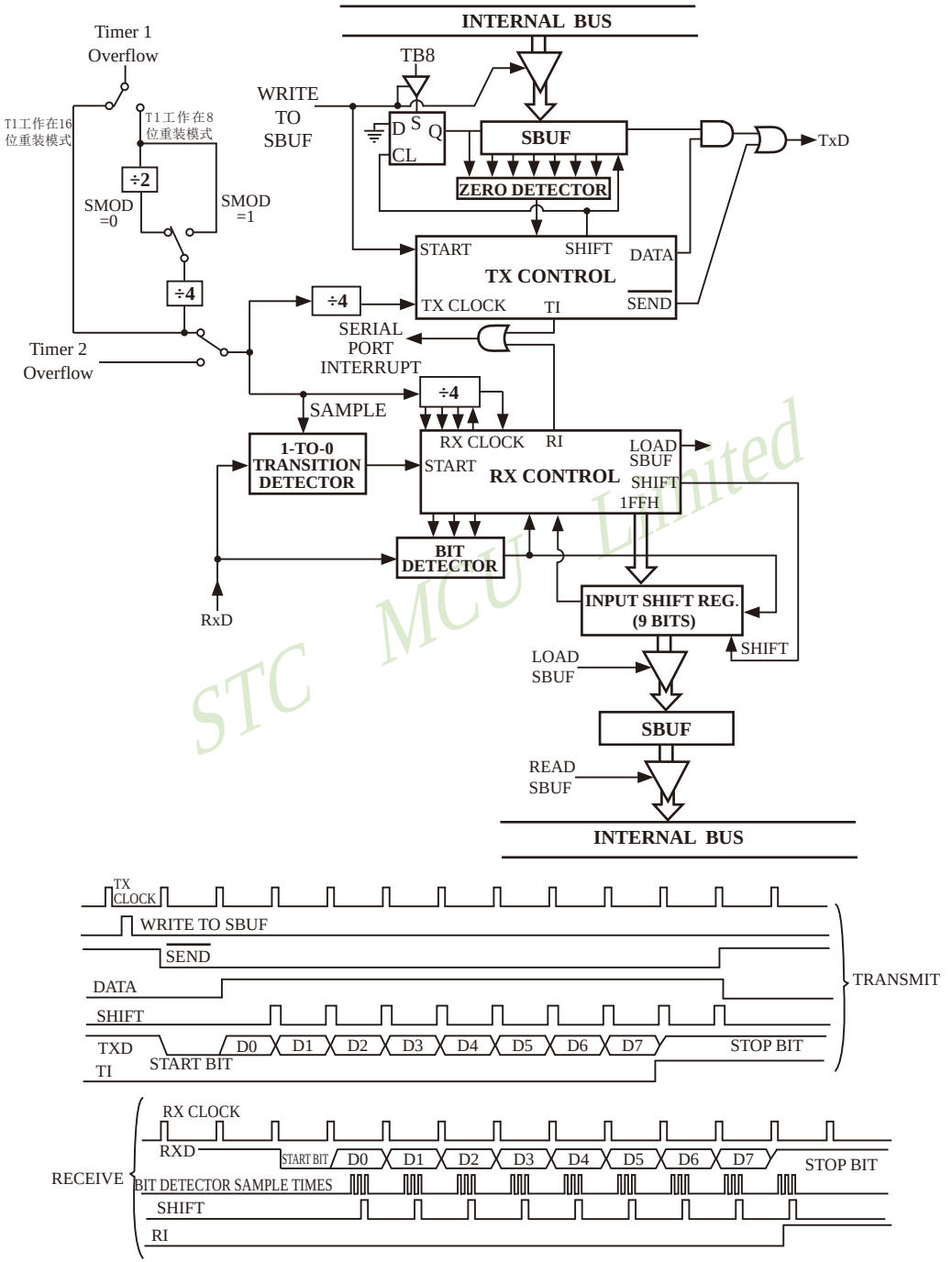


图8-2　串行口模式1功能结构示意图及接收/发送时序图

STC MCU Limited

串行通信模式1的波特率是可变的，可变的波特率由定时器/计数器1或定时器2产生，优先选择定时器2产生波特率。

STC创新设计，请不要抄袭，再抄袭就很无耻了

当串行口1用定时器2作为其波特率发生器时，

串行口1的波特率=(定时器T2的溢出率)/4.

（注意：此时波特率也与SMOD无关。）

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率 = SYSclk / (65536 - [RL_TH2,RL_TL2])；

即此时，串行口1的波特率=SYSclk / (65536 - [RL_TH2,RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2,RL_TL2])；

即此时，串行口1的波特率=SYSclk / 12 / (65536 - RL_TH2,RL_TL2]) / 4

RL_TH2是T2H的自动重装载寄存器，RL_TL2是T2L的自动重装载寄存器。

当串行口1用定时器1作为其波特率发生器且定时器1工作于模式0(16位自动重装载模式)时，

串行口1的波特率=(定时器1的溢出率)/4.

（注意：此时波特率与SMOD无关。）

当定时器1工作于模式0（16位自动重装载模式）且T1x12 = 0时，

定时器1的溢出率 = SYSclk/12/(65536 - [RL_TH1,RL_TL1]) ;

即此时，串行口1的波特率=SYSclk/12/(65536 - [RL_TH1,RL_TL1]) / 4

当定时器1工作于模式0（16位自动重装载模式）且T1x12 = 1时,

定时器1的溢出率 = SYSclk / (65536 - [RL_TH1,RL_TL1])

即此时，串行口1的波特率=SYSclk / (65536 - [RL_TH1,RL_TL1]) / 4

RL_TH1是TH1的自动重装载寄存器，RL_TL1是TL1的自动重装载寄存器。

当串行口1用定时器1作为其波特率发生器且定时器1工作于模式2(8位自动重装模式)时，

串行口1的波特率=(2^{SMOD}/32)×(定时器1的溢出率).

当定时器1工作于模式2（8位自动重装模式）且T1x12 = 0时，

定时器1的溢出率 = SYSclk / 12 / (256 - TH1);

即此时，串行口1的波特率=(2^{SMOD}/32)×SYSclk / 12 / (256 - TH1)

当定时器1工作于模式2（8位自动重装模式）且T1x12 = 1时,

定时器1的溢出率 = SYSclk / (256 - TH1)

即此时，串行口1的波特率=(2^{SMOD}/32)×SYSclk / (256 - TH1)

STC MCU Limited

8.2.3 串行口1工作模式2：9位UART，波特率固定(建议不学习)

当SM0、SM1两位为10时，串行口1工作在模式2。串行口1工作模式2为9位数据异步通信UART模式，其一帧的信息由11位组成：1位起始位，8位数据位(低位在先)，1位可编程位(第9位数据)和1位停止位。发送时可编程位(第9位数据)由SCON中的TB8提供，可软件设置为1或0，或者可将PSW中的奇/偶校验位P值装入TB8(TB8既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位)。接收时第9位数据装入SCON的RB8。TxD/P3.1为发送端口，RxD/P3.0为接收端口，以全双工模式进行接收/发送。

模式2的波特率为：

串行通信模式2波特率=$2^{SMOD}/64\times$(SYSclk系统工作时钟频率)

上述波特率可通过软件对PCON中的SMOD位进行设置，当SMOD=1时，选择SYSclk/32；当SMOD=0时，选择SYSclk/64，故而称SMOD为波特率加倍位。可见，模式2的波特率基本上是固定的。

图8-3为串行通信模式2的功能结构示意图及其接收/发送时序图。

由图8-3可知，模式2和模式1相比，除波特率发生源略有不同，发送时由TB8提供给移位寄存器第9数据位不同外，其余功能结构均基本相同，其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

- RI=0
- SM2=0或者SM2=1，并且接收到的第9数据位RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入SBUF和RB8中，并置位RI=1，向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位RI。无论上述条件满足与否，接收器又重新开始检测RxD输入端口的跳变信息，接收下一帧的输入信息。

在模式2中，接收到的停止位与SBUF、RB8和RI无关。

通过软件对SCON中的SM2、TB8的设置以及通信协议的约定，为多机通信提供了方便。

STC MCU Limited

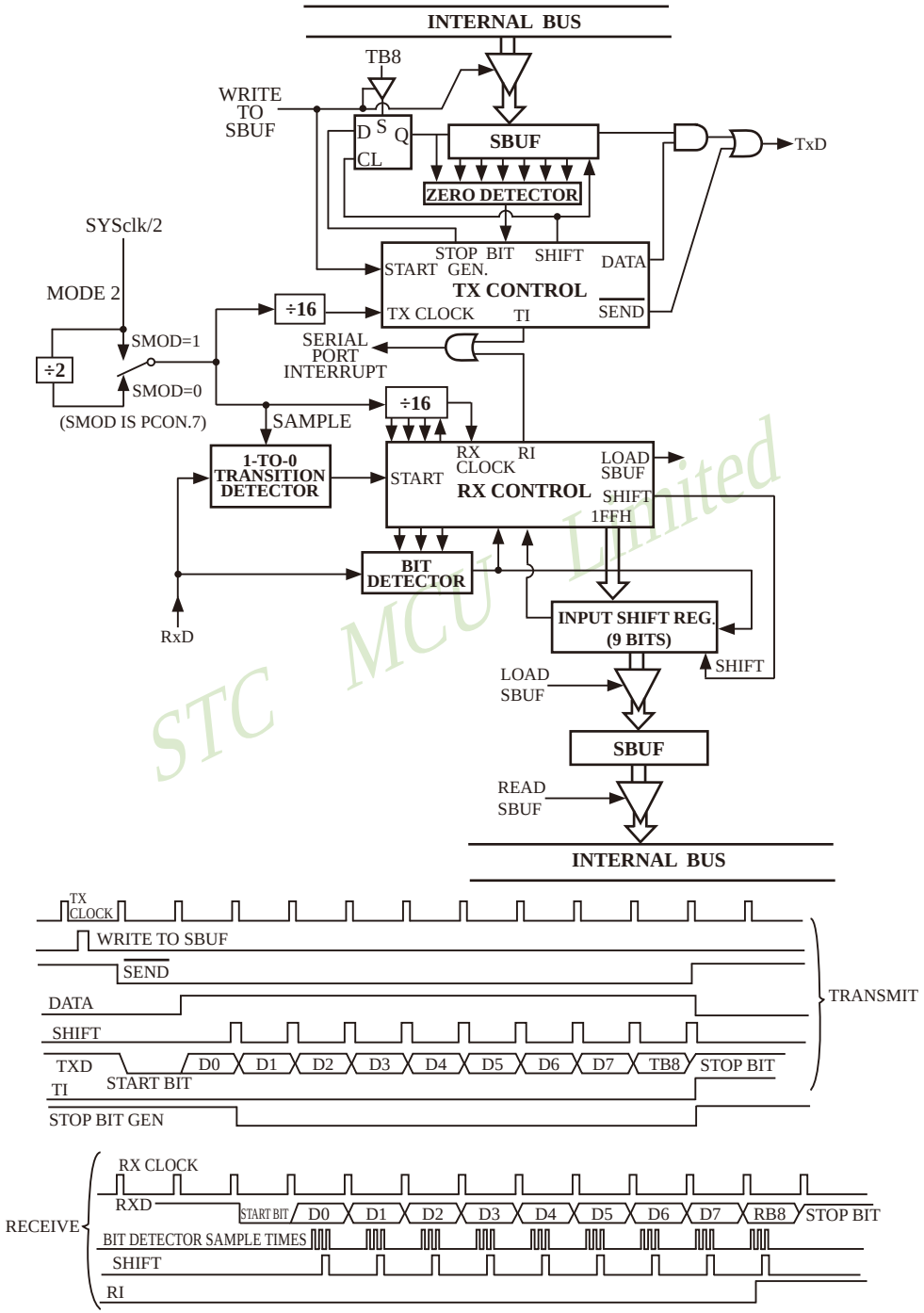


图8-3　串行口模式2功能结构示意图及接收/发送时序图

STC MCU Limited

8.2.4 串行口1工作模式3：9位UART，波特率可变

当SM0、SM1两位为11时，串行口1工作在模式3。**串行通信模式3**为9位数据异步通信UART模式，**其一帧的信息由**11位组成：1位起始位，8位数据位(低位在先)，1位可编程位(第9位数据)和1位停止位。发送时可编程位(第9位数据)由SCON中的TB8提供，可软件设置为1或0，或者可将PSW中的奇/偶校验位P值装入TB8(TB8既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位)。接收时第9位数据装入SCON的RB8。TxD/P3.1为发送端口，RxD/P3.0为接收端口，以全双工模式进行接收/发送。

图8-4为串行口工作模式3的功能结构示意图及其接收/发送时序图。

由图8-4可知，模式3和模式1相比，除发送时由TB8提供给移位寄存器第9数据位不同外，其余功能结构均基本相同，其接收**‘发送操作过程及时序也基本相同。**

当接收器接收完一帧信息后必须同时满足下列条件：

- RI=0
- SM2=0或者SM2=1，并且接收到的第9数据位RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入SBUF和RB8中，并置位RI=1，向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位RI。无论上述条件满足与否，接收器又重新开始检测RxD输入端口的跳变信息，接收下一帧的输入信息。

在模式3中，接收到的停止位与SBUF、RB8和RI无关。

通过软件对SCON中的SM2、TB8的设置以及通信协议的约定，为多机通信提供了方便。

串行通信模式3的波特率也是可变的，可变的波特由定时器/计数器1或定时器2产生。

STC MCU Limited

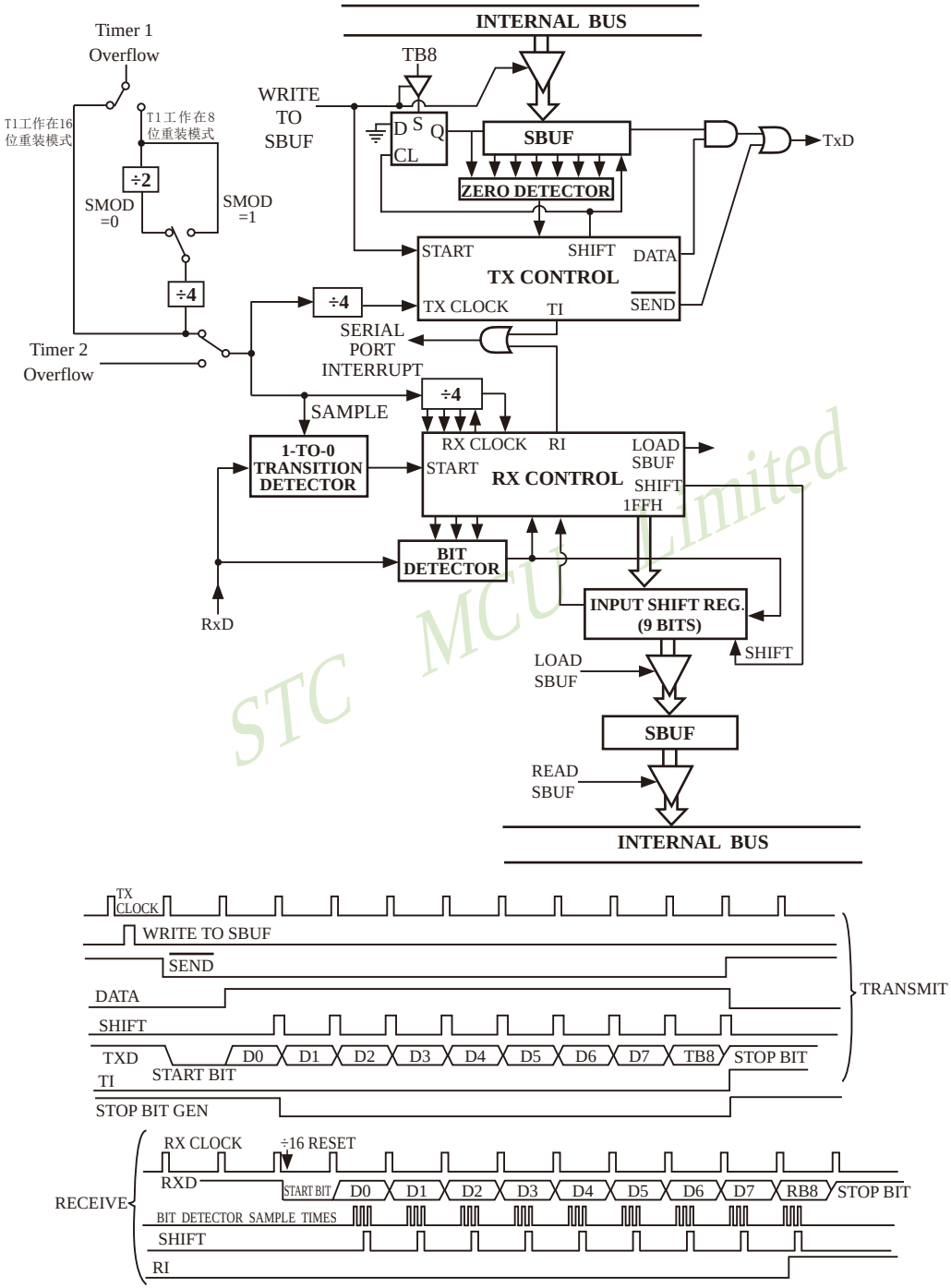


图8-4　串行口模式3功能结构示意图及接收/发送时序图

STC MCU Limited

模式3的波特率（优先选择定时器2产生波特率）为：

当串行口1用定时器2作为其波特率发生器时，

串行口1的波特率=(定时器T2的溢出率)/4.

（注意：此时波特率也与SMOD无关。）

STC创新设计，请不要抄袭，再抄袭就很无耻了

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率 = SYSclk / (65536 - [RL_TH2,RL_TL2])；

即此时，串行口1的波特率=SYSclk / (65536 - [RL_TH2,RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [[RL_TH2,RL_TL2])；

即此时，串行口1的波特率=SYSclk / 12 / (65536 - [RL_TH2,RL_TL2]) / 4

RL_TH2是T2H的自动重装载寄存器，RL_TL2是T2L的自动重装载寄存器。

当串行口1用定时器1作为其波特率发生器且定时器1工作于模式0(16位自动重装载模式)时，

串行口1的波特率=(定时器1的溢出率)/4.

（注意：此时波特率与SMOD无关。）

当定时器1工作于模式0（16位自动重装载模式）且T1x12 = 0时，

定时器1的溢出率 = SYSclk/12/(65536 - [RL_TH1,RL_TL1]) ;

即此时，串行口1的波特率=SYSclk/12/(65536 - [RL_TH1, RL_TL1]) / 4

当定时器1工作于模式0（16位自动重装载模式）且T1x12 = 1时,

定时器1的溢出率 = SYSclk / (65536 - [RL_TH1,RL_TL1])

即此时，串行口1的波特率=SYSclk / (65536 - [RL_TH1, RL_TL1]) / 4

RL_TH1是TH1的自动重装载寄存器，RL_TL1是TL1的自动重装载寄存器。

当串行口1用定时器1作为其波特率发生器且定时器1工作于模式2(8位自动重装模式)时，

串行口1的波特率=(2^{SMOD}/32)×(定时器1的溢出率).

当定时器1工作于模式2（8位自动重装模式）且T1x12 = 0时，

定时器1的溢出率 = SYSclk / 12 / (256 - TH1);

即此时，串行口1的波特率=(2^{SMOD}/32)×SYSclk / 12 / (256 - TH1)

当定时器1工作于模式2（8位自动重装模式）且T1x12 = 1时,

定时器1的溢出率 = SYSclk / (256 - TH1)

即此时，串行口1的波特率=(2^{SMOD}/32)×SYSclk / (256 - TH1)

可见，模式3和模式1一样，其波特率可通过软件对定时器/计数器1或定时器2的设置进行波特率的选择，是可变的。

STC-MCU Limited

8.3 串行口1的波特率设置

——串口1和串口2的波特率相同时,串口1和串口2可共享T2作波特率发生器

STC15W4K60S4系列单片机串行口1的波特率随所选工作模式的不同而异，对于工作模式0和模式2，其波特率与系统时钟频率SYSclk和PCON中的波特率选择位SMOD有关，而模式1和模式3的波特率除与SYSclk和PCON位有关外，还与定时器/计数器1或定时器2设置有关。通过对定时器/计数器1或定时器2的设置，可选择不同的波特率，所以这种波特率是可变的。建议用户优先选择定时器2作为串行口1的波特率发生器。

说明：当串口1和串口2的波特率相同时，串口1和串口2可以共享波特率发生器，此时建议用户选择定时器T2作为串口1的波特率发生器；当串口1和串口2的波特率不同时，才建议选择定时器T1作为串口1的波特率发生器(因串口2固定使用定时器T2作波特率发生器)。

串行通信模式0，其波特率与系统时钟频率SYSclk有关 。

当模式0的通信速度设置位UART_M0x6/AUXR.5 = 0时，其波特率 = SYSclk/12。

当模式0的通信速度设置位UART_M0x6/AUXR.5 = 1时，其波特率 = SYSclk/2。

一旦SYSclk选定且UART_M0x6/AUXR.5设置好，则串行通信工作模式0的波特率固定不变。

串行通信工作模式2，其波特率除与SYSclk有关外，还与SMOD位有关。

其基本表达式为：串行通信模式2波特率=$2^{SMOD}/64\times$(SYSclk系统工作时钟频率)

当SMOD=1时，波特率=2/64(SYSclk)=1/32(SYSclk)；

当SMOD=0时，波特率=1/64(SYSclk)。

当SYSclk选定后，通过软件设置PCON中的SMOD位，可选择两种波特率。所以，这种模式的波特率基本固定。

串行通信模式1和3，其波特率是可变的(建议用户优先选择定时器T2作为串口1的波特率发生器)：

当串行口1用定时器2作为其波特率发生器时，

串行口1的波特率=(定时器T2的溢出率)/4.

（注意：此时波特率也与SMOD无关。）

STC创新设计，请不要抄袭，再抄袭就很无耻了

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率 = SYSclk / (65536 - [RL_TH2,RL_TL2])；

即此时，串行口1的波特率=SYSclk / (65536 - [RL_TH2,RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2,RL_TL2])；

即此时，串行口1的波特率=SYSclk / 12 / (65536 - [RL_TH2,RL_TL2]) / 4

STC MCU Limited

当串行口1用定时器1作为其波特率发生器且定时器1工作于模式0(16位自动重装载模式)时，

串行口1的波特率=(定时器1的溢出率)/4.

（注意：此时波特率与SMOD无关。）

当定时器1工作于模式0（16位自动重装载模式）且T1x12 = 0时，

定时器1的溢出率 = SYSclk/12/(65536 - [RL_TH1,RL_TL1]) ;

即此时，串行口1的波特率=SYSclk/12/(65536 - [RL_TH1,RL_TL1]) / 4

当定时器1工作于模式0（16位自动重装载模式）且T1x12 = 1时,

定时器1的溢出率 = SYSclk / (65536 - [RL_TH1,RL_TL1])

即此时，串行口1的波特率=SYSclk / (65536 - [RL_TH1,RL_TL1]) / 4

当串行口1用定时器1作为其波特率发生器且定时器1工作于模式2(8位自动重装模式)时，

串行口1的波特率=(2^{SMOD}/32)×(定时器1的溢出率).

当定时器1工作于模式2（8位自动重装模式）且T1x12 = 0时，

定时器1的溢出率 = SYSclk / 12 / (256 - TH1);

即此时，串行口1的波特率=(2^{SMOD}/32)×SYSclk / 12 / (256 - TH1)

当定时器1工作于模式2（8位自动重装模式）且T1x12 = 1时,

定时器1的溢出率 = SYSclk / (256 - TH1)

即此时，串行口1的波特率=(2^{SMOD}/32)×SYSclk / (256 - TH1)

STC MCU Limited

通过对定时器1和定时器2的设置，可灵活地选择不同的波特率。在实际应用中多半选用串行模式1或串行模式3。显然，为选择波特率，关键在于定时器1和定时器2的溢出率的计算。SMOD的选择，只需根据需要执行下列指令就可实现SMOD=0或1；

```
MOV     PCON，#00H              ；使SMOD=0
MOV     PCON，#80H              ；使SMOD=1
```

SMOD只占用电源控制寄存器PCON的最高一位，其他各位的具体设置应根据实际情况而定。

当用户选择定时器/计数器1作波特率发生器时，为选择波特率，关键在于定时器/计数器1的溢出率。下面介绍如何计算定时器/计数器1的溢出率。

定时器/计数器1的溢出率定义为：单位时间（秒）内定时器/计数器1回0溢出的次数，即定时器/计数器1的溢出率=定时器/计数器1的溢出次数/秒。

STC15W4K60S4系列单片机设有3个定时器/计数器，定时器/计数器1具有4种工作方式,而常选用定时器/计数器1的工作方式0（16位自动重装载模式）及工作方式2（8位自动重装）作为波特率的溢出率。

以定时器/计数器1工作于定时模式的工作方式2（8位自动重装）为例：设置定时器/计数器1工作于定时模式的工作方式2（8位自动重装），TL1的计数输入来自于SYSclk经12分频或不分频(由T1x12/AUXR.6确定是12分频还是不分频)的脉冲。当T1x12/AUXR.6=0时，单片机工作在12T模式，TL1的计数输入来自于SYSclk经12分频的脉冲；当T1x12/AUXR.6=1时，单片机工作在1T模式，TL1的计数输入来自于SYSclk不经过分频的脉冲。可见，定时器/计数器1的溢出率与SYSclk和自动重装值N有关，SYSclk越大，特别是N越大，溢出率也就越高。例如：当N=FFN，则每隔一个时钟即溢出一次（极限情况）；若N=00H，则需每隔256个时钟才溢出一次；当SYSclk=6MHz且T1x12/AUXR.6=0时，一个时钟为2μs, 当SYSclk=6MHz且T1x12/AUXR.6=1时，一个时钟约为0.167μs(快12倍)。SYSclk=12MHz且T1x12/AUXR.6=0时，则一个时钟为1μs, 当SYSclk=6MHz且T1x12/AUXR.6=1时，一个时钟约为0.083μs(快12倍)。

对于一般情况下，

当T1x12/AUXR.6=0时，定时器/计数器1溢出一次所需的时间为: $(2^8-N)\times 12$时钟$=(2^8-N)\times 12\times\frac{1}{SYSclk}$

当T1x12/AUXR.6=1时，定时器/计数器1溢出一次所需的时间为: $(2^8-N)\times 1$时钟$=(2^8-N)\times\frac{1}{SYSclk}$

于是得定时器/计数器每秒溢出的次数，即

当T1x12/AUXR.6=0时，定时器/计数器1的溢出率=SYSclk/12×(2^8-N) (次/秒)

当T1x12/AUXR.6=1时，定时器/计数器1的溢出率=SYSclk×(2^8-N) (次/秒)

式中SYSclk为系统时钟频率，N为再装入时间常数。

显然，选用定时器/计数器2作波特率的溢出率也一样。选用不同工作方式所获得波特率的范围不同。因为不同方式的计数位数不同，N取值范围不同，且计数方式较复杂。

现以定时器/计数器1工作于方式2（8位自动重装模式）为例，

设：T1x12/AUXR.6=0, SYSclk=6MHz, N=FFH,

定时器/计数器1工作于方式2的溢出率为$6\times10^6/\{12\times(256-255)\}=0.5\times10^6$(次/秒)；

设：T1x12/AUXR.6=0, SYSclk=12MHz, N=FFH,

定时器/计数器1工作于方式2的溢出率 = 1×10^6（次/秒）；

设：T1x12/AUXR.6=0, SYSclk=12MHz, N=00H,

定时器/计数器1工作于方式2的溢出率 = $12\times10^6/12\times256\approx3906$（次/秒）

设：T1x12/AUXR.6=1, SYSclk=6MHz, N=FFH,

定时器/计数器1工作于方式2的溢出率为$6\times10^6/(256-255)=6\times10^6$(次/秒)；

设：T1x12/AUXR.6=1, SYSclk=12MHz, N=00H,

定时器/计数器1工作于方式2的溢出率 = $12\times10^6/256=46875$（次/秒）

STC MCU Limited

下表给出各种常用波特率与定时器/计数器1各参数之间的关系。

常用波特率与定时器/计数器1各参数关系（T1x12/AUXR.6=0）

常用波特率	系统时钟频率(MHz)	SMOD	定时器1		
			C/$\overline{T}$	方式	重新装入值
方式0 MAX：1M	12	×	×	×	×
方式2 MAX：375K	12	1	×	×	×
方式1和3　62.5K	12	1	0	2	FFH
19.2K	11.059	1	0	2	FDH
9.6K	11.059	0	0	2	FDH
4.8K	11.059	0	0	2	FAH
2.4K	11.059	0	0	2	F4H
1.2K	11.059	0	0	2	F8H
137.5	11.986	0	0	2	1DH
110	6	0	0	2	72H
110	12	0	0	1	FFFBH

设置波特率的初始化程序段如下：

```
        ⋮
MOV   TMOD,    #20H     ; 设置定时器/计数器1定时、工作方式2
MOV   TH1,     #xxH     ; 设置定时常数N
MOV   TL1,     #xxH     ;
SETB  TR1               ; 启动定时器/计数器1
MOV   PCON,    #80H     ; 设置SMOD=1
MOV   SCON,    #50H     ; 设置串行通信方式1
        ⋮
```

执行上述程序段后，即可完成对定时器/计数器1的操作方式及串行通信的工作方式和波特率的设置。

由于用其他工作方式设置波特率计算方法较复杂，一般应用较少，故不一一论述。

当用户选择定时器2作波特率发生器时，为选择波特率，关键在于定时器2的溢出率。当用户选择定时器2作波特率发生器时，**定时器/计数器1可以释放出来作为定时器/计数器/时钟输**出使用。

用户在程序中如何具体使用串口1和**定时器T2**

1. 设置串口1 的工作模式，SCON 寄存器中的SM0 和SM1 两位决定了串口1 的4 种工作模式。
2. 设置串口1 的波特率,使用**定时器2寄存器 T2H及T2L**
3. 设置寄存器AUXR中的位T2x12/AUXR.2,确定定时器2速度是1T还是12T
4. 启动**定时器2，让T2R位为1，T2H/T2L 定时器2寄存器就立即开始计数。**
5. 设置串口1的中断优先级，及打开中断相应的控制位是:
   PS,ES,EA
6. 如要串口1接收，将REN置1即可
   如要串口1发送，将数据送入SBUF即可，
   接收完成标志RI,发送完成标志TI,要由软件清0。

STC MCU Limited

当串口1工作在模式1和模式3时，**计算相应的波特率需要设置的重装载数，结果送入T2H/T2L寄存器，计算自动重装数 RELOAD:**

1. 计算 RELOAD
   计算公式： RELOAD = 65536 - INT(SYSclk/Baud0/4 + 0.5)

计算出的RELOAD 数直接送T2H/T2L寄存器

式中: INT() 表示取整运算即舍去小数, 在式中加 0.5 可以达到四舍五入的目的
SYSclk = 晶振频率
Baud0 = 标准波特率

2. 设置AUXR.2/T2x12=1,定时器2工作在1T模式
3. 计算用 RELOAD 产生的波特率:
   Baud = SYSclk/(65536 - RELOAD)/4
4. 计算误差
error = (Baud - Baud0)/Baud0 * 100%
5. 如果误差绝对值 > 3% 要更换波特率或者更换晶体频率, 重复步骤 1-4

例: SYSclk = 22.1184MHz, Baud0 = 57600

1. RELOAD = 65536 - INT(22118400/57600/4 + 0.5)
   = 65536 - INT(96.5)
   = 65536 - 96
   = 65440
   = 0FFA0H
2. 设置AUXR.2/T2x12=1,定时器2工作在1T模式
3. Baud = 22118400/(65536-65440)/4
   = 57600
4. 误差等于零

例: SYSclk = 12MHz, Baud0 = 57600

1. RELOAD = 65536 - INT(12000000/57600/4 + 0.5)
= 65536 - INT(52.0833 + 0.5)
= 65536 - INT(52.5833)
= 65536 - 52
= 65484
= 0FFCCH
2. 设置AUXR.2/T2x12=1,定时器2工作在1T模式
3. Baud = 12000000/(66536-65484)/4
= 57692
4. error = (57692 - 57600)/57600 * 100%
   = 0.16%

STC MCU Limited

8.4 串行口1的测试程序(C和汇编)

8.4.1 定时器2作串口1波特率发生器的测试程序(C和汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ------------------------------------------------------------------------------ */
/* --- STC15F2K60S2 系列 定时器2用作串口1的波特率发生器举例---------------------- */
/* --- 研发顾问QQ：800003751----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 --------------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ---------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----              */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----              */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define  FOSC    18432000L                    //系统频率
#define  BAUD    115200                       //串口波特率

#define  NONE_PARITY        0                 //无校验
#define  ODD_PARITY         1                 //奇校验
#define  EVEN_PARITY        2                 //偶校验
#define  MARK_PARITY        3                 //标记校验
#define  SPACE_PARITY       4                 //空白校验

#define PARITYBIT EVEN_PARITY                 //定义校验位

sfr      AUXR    =    0x8e;                          //辅助寄存器
sfr      T2H     =    0xd6;                          //定时器2高8位
sfr      T2L     =    0xd7;                          //定时器2低8位

sbit     P22     =    P2^2;

bit      busy;

void SendData(BYTE dat);
void SendString(char *s);
void main()
```

STC MCU Limited

```
{
#if (PARITYBIT == NONE_PARITY)
	SCON = 0x50;				//8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
	SCON = 0xda;				//9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
	SCON = 0xd2;				//9位可变波特率,校验位初始为0
#endif

	T2L = (65536 - (FOSC/4/BAUD));		//设置波特率重装值
	T2H = (65536 - (FOSC/4/BAUD))>>8;
	AUXR = 0x14;				//T2为1T模式, 并启动定时器2
	AUXR |= 0x01;				//选择定时器2为串口1的波特率发生器
	ES = 1;					//使能串口1中断
	EA = 1;

	SendString("STC15F2K60S2\r\nUart Test !\r\n");
	while(1);
}

/*----------------------------
UART 中断服务程序
-----------------------------*/
void Uart() interrupt 4 using 1
{
	if (RI)
	{
		RI = 0;				//清除RI位
		P0 = SBUF;			//P0显示串口数据
		P22 = RB8;			//P2.2显示校验位
	}
	if (TI)
	{
		TI = 0;				//清除TI位
		busy = 0;			//清忙标志
	}
}

/*----------------------------
发送串口数据
-----------------------------*/
void SendData(BYTE dat)
{
	while (busy);				//等待前面的数据发送完成
	ACC = dat;				//获取校验位P (PSW.0)
	if (P)					//根据P来设置校验位
	{
	#if (PARITYBIT == ODD_PARITY)
		TB8 = 0;			//设置校验位为0
```

STC MCU Limited

```
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 1;                    //设置校验位为1
        #endif
        }
        else
        {
        #if (PARITYBIT == ODD_PARITY)
            TB8 = 1;                    //设置校验位为1
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 0;                    //设置校验位为0
        #endif
        }
        busy = 1;
        SBUF = ACC;                     //写数据到UART数据寄存器
}

/*---------------------------
发送字符串
---------------------------*/
void SendString(char *s)
{
        while (*s)                      //检测字符串结束标志
        {
            SendData(*s++);             //发送当前字符
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ------------------------------------------------------------------------------ */
/* --- STC15F2K60S2 系列 定时器2用作串口1的波特率发生器举例---------------------------------------- */
/* --- 研发顾问QQ：800003751-------------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 -------------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com --------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*---------------------------------------------------------------------------------------------------- */

//本示例在Keil开发环境下请选择Intel的8052芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz

#define   NONE_PARITY   0                       //无校验
#define   ODD_PARITY    1                       //奇校验
#define   EVEN_PARITY   2                       //偶校验
#define   MARK_PARITY   3                       //标记校验
#define   SPACE_PARITY  4                       //空白校验

#define   PARITYBIT     EVEN_PARITY             //定义校验位

//-----------------------------------------

AUXR    EQU     08EH                            //辅助寄存器
T2H     DATA    0D6H                            //定时器2高8位
T2L     DATA    0D7H                            //定时器2低8位

//-----------------------------------------
BUSY    BIT     20H.0                           //忙标志位
//-----------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     0023H
        LJMP    UART_ISR
//-----------------------------------------
        ORG     0100H
MAIN:
        CLR     BUSY
        CLR     EA
        MOV     SP,     #3FH

#if (PARITYBIT == NONE_PARITY)
        MOV     SCON,   #50H                    //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
```

STC MCU Limited

```
        MOV     SCON,   #0DAH                   //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     SCON,   #0D2H                   //9位可变波特率,校验位初始为0
#endif

//---------------------------------
        MOV     T2L,    #0D8H                   //设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H                    //T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H                    //选择定时器2为串口1的波特率发生器
         SETB   ES                              //使能串口中断
        SETB    EA

        MOV     DPTR,   #TESTSTR                //发送测试字符串
        LCALL   SENDSTRING

        SJMP    $
;------------------------------------------
TESTSTR:
        DB  "STC15F2K60S2 Uart1 Test !",0DH,0AH,0

;/*---------------------------
;UART 中断服务程序
;---------------------------*/
UART_ISR:
        PUSH    ACC
        PUSH    PSW
        JNB     RI,     CHECKTI                 //检测RI位
        CLR     RI                              //清除RI位
        MOV     P0,     SBUF                    //P0显示串口数据
        MOV     C,      RB8
        MOV     P2.2,   C                       //P2.2显示校验位
CHECKTI:
        JNB     TI,     ISR_EXIT                //检测TI位
        CLR     TI                              //清除TI位
        CLR     BUSY                            //清忙标志
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

;/*---------------------------
;发送串口数据
;---------------------------*/
SENDDATA:
        JB      BUSY,   $                       //等待前面的数据发送完成
        MOV     ACC,    A                       //获取校验位P (PSW.0)
        JNB     P,      EVEN1INACC              //根据P来设置校验位
```

STC MCU Limited

```
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        CLR     TB8                     //设置校验位为0
#elif (PARITYBIT == EVEN_PARITY)
        SETB    TB8                     //设置校验位为1
#endif
        SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        SETB    TB8                     //设置校验位为1
#elif (PARITYBIT == EVEN_PARITY)
        CLR     TB8                     //设置校验位为0
#endif
PARITYBITOK:                            //校验位设置完成
        SETB    BUSY
        MOV     SBUF,   A               //写数据到UART数据寄存器
        RET

;/*---------------------------
;发送字符串
//---------------------------*/
SENDSTRING:
        CLR     A
        MOVC    A,      @A+DPTR         //读取字符
        JZ      STRINGEND               //检测字符串结束标志
        INC     DPTR                    //字符串地址+1
        LCALL   SENDDATA                //发送当前字符
        SJMP    SENDSTRING
STRINGEND:
        RET
//-----------------------------------------
        END
```

STC MCU Limited

8.4.2 定时器1模式0(16位自动重装载)作串口1波特率发生器程序(C和汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器1用作串口1的波特率发生器举例---------------------------------------- */
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define  FOSC    18432000L                          //系统频率
#define  BAUD    115200                             //串口波特率

#define  NONE_PARITY     0                          //无校验
#define  ODD_PARITY      1                          //奇校验
#define  EVEN_PARITY     2                          //偶校验
#define  MARK_PARITY     3                          //标记校验
#define  SPACE_PARITY    4                          //空白校验

#define PARITYBIT EVEN_PARITY                       //定义校验位

sfr      AUXR    =       0x8e;                      //辅助寄存器

sbit     P22     =       P2^2;

bit      busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
        SCON = 0x50;                                //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        SCON = 0xda;                                //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        SCON = 0xd2;                                //9位可变波特率,校验位初始为0
#endif
```

```
	AUXR = 0x40;                        //定时器1为1T模式
	TMOD = 0x00;                        //定时器1为模式0(16位自动重载)
	TL1 = (65536 - (FOSC/4/BAUD));      //设置波特率重装值
	TH1 = (65536 - (FOSC/4/BAUD))>>8;
	TR1 = 1;                            //定时器1开始启动
	ES = 1;                             //使能串口中断
	EA = 1;

	SendString("STC15F2K60S2\r\nUart Test !\r\n");
	while(1);
}

/*----------------------------
UART 中断服务程序
-----------------------------*/
void Uart() interrupt 4 using 1
{
	if (RI)
	{
		RI = 0;                     //清除RI位
		P0 = SBUF;                  //P0显示串口数据
		P22 = RB8;                  //P2.2显示校验位
	}
	if (TI)
	{
		TI = 0;                     //清除TI位
		busy = 0;                   //清忙标志
	}
}

/*----------------------------
发送串口数据
-----------------------------*/
void SendData(BYTE dat)
{
	while (busy);                       //等待前面的数据发送完成
	ACC = dat;                          //获取校验位P (PSW.0)
	if (P)                              //根据P来设置校验位
	{
	#if (PARITYBIT == ODD_PARITY)
		TB8 = 0;                    //设置校验位为0
	#elif (PARITYBIT == EVEN_PARITY)
		TB8 = 1;                    //设置校验位为1
	#endif
	}
	else
	{
	#if (PARITYBIT == ODD_PARITY)
		TB8 = 1;                    //设置校验位为1
```

STC MCU Limited

```
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 0;                            //设置校验位为0
        #endif
        }
        busy = 1;
        SBUF = ACC;                             //写数据到UART数据寄存器
}

/*---------------------------
发送字符串
---------------------------*/
void SendString(char *s)
{
        while (*s)                              //检测字符串结束标志
        {
            SendData(*s++);                     //发送当前字符
        }
}
```

2. 汇编程序：

```
/*------------------------------------------------------------------------------------------------------ */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器1用作串口1的波特率发生器举例---------------------- */
/* --- 研发顾问QQ：  800003751----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----              */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----              */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-----------*/
/*------------------------------------------------------------------------------------------------------ */

//假定测试芯片的工作频率为18.432MHz

#define   NONE_PARITY   0                   //无校验
#define   ODD_PARITY    1                   //奇校验
#define   EVEN_PARITY   2                   //偶校验
#define   MARK_PARITY   3                   //标记校验
#define   SPACE_PARITY  4                   //空白校验

#define   PARITYBIT EVEN_PARITY             //定义校验位

//---------------------------------------

AUXR    EQU     08EH                        //辅助寄存器

BUSY    BIT     20H.0                       //忙标志位

//---------------------------------------
```

STC MCU Limited

```
        ORG     0000H
        LJMP    MAIN

        ORG     0023H
        LJMP    UART_ISR
//----------------------------------------
        ORG     0100H
MAIN:
        CLR     BUSY
        CLR     EA
        MOV     SP,     #3FH

#if (PARITYBIT == NONE_PARITY)
        MOV     SCON,   #50H            //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        MOV     SCON,   #0DAH           //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     SCON,   #0D2H           //9位可变波特率,校验位初始为0
#endif

//---------------------------------
        MOV     AUXR,   #40H            //定时器1为1T模式
        MOV     TMOD,   #00H            //定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H           //设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                     //定时器1开始运行
        SETB    ES                      //使能串口中断
        SETB    EA

        MOV     DPTR,   #TESTSTR        //发送测试字符串
        LCALL   SENDSTRING

        SJMP    $
;----------------------------------------
TESTSTR:
        DB  "STC15F2K60S2 Uart1 Test !",0DH,0AH,0

;/*----------------------------
;UART 中断服务程序
;----------------------------*/
UART_ISR:
        PUSH    ACC
        PUSH    PSW
        JNB     RI,     CHECKTI         //检测RI位
        CLR     RI                      //清除RI位
        MOV     P0,     SBUF                    //P0显示串口数据
        MOV     C,      RB8
        MOV     P2.2,   C                       //P2.2显示校验位
```

STC MCU Limited

```
CHECKTI:
        JNB     TI,     ISR_EXIT            //检测TI位
        CLR     TI                          //清除TI位
        CLR     BUSY                        //清忙标志
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

;/*----------------------------
;发送串口数据
;----------------------------*/
SENDDATA:
        JB      BUSY,   $                   //等待前面的数据发送完成
        MOV     ACC,    A                   //获取校验位P (PSW.0)
        JNB     P,      EVEN1INACC          //根据P来设置校验位
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        CLR     TB8                         //设置校验位为0
#elif (PARITYBIT == EVEN_PARITY)
        SETB    TB8                         //设置校验位为1
#endif
        SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        SETB    TB8                         //设置校验位为1
#elif (PARITYBIT == EVEN_PARITY)
        CLR     TB8                         //设置校验位为0
#endif
PARITYBITOK:                                //校验位设置完成
        SETB    BUSY
        MOV     SBUF,   A                   //写数据到UART数据寄存器
        RET

;/*----------------------------
;发送字符串
//----------------------------*/
SENDSTRING:
        CLR     A
        MOVC    A,      @A+DPTR             //读取字符
        JZ      STRINGEND                   //检测字符串结束标志
        INC     DPTR                        //字符串地址+1
        LCALL   SENDDATA                    //发送当前字符
        SJMP    SENDSTRING
STRINGEND:
        RET
//-----------------------------------------
        END
```

STC MCU Limited

8.4.3 定时器1模式2(8位自动重装载)作串口1波特率发生器程序(建议不学)

1. C程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器1用作串口1的波特率发生器举例---------------------- */
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*---------------------------------------------------------------------------------------------------- */


//假定测试芯片的工作频率为18.432MHz#include "reg51.h"

#include "intrins.h"

typedef unsigned char        BYTE;
typedef unsigned int         WORD;

#define   FOSC    18432000L                 //系统频率
#define   BAUD    115200                    //串口波特率

#define   NONE_PARITY       0       //无校验
#define   ODD_PARITY        1       //奇校验
#define   EVEN_PARITY       2       //偶校验
#define   MARK_PARITY       3       //标记校验
#define   SPACE_PARITY      4       //空白校验

#define PARITYBIT EVEN_PARITY   //定义校验位

sfr       AUXR    =         0x8e;             //辅助寄存器

sbit      P22     =         P2^2;

bit busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
        SCON = 0x50;                        //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        SCON = 0xda;                        //9位可变波特率,校验位初始为1
```

STC MCU Limited

```
#elif (PARITYBIT ==     SPACE_PARITY)
        SCON    =       0xd2;                   //9位可变波特率,校验位初始为0
#endif

        AUXR    =       0x40;                   //定时器1为1T模式
        TMOD    =       0x20;                   //定时器1为模式2(8位自动重载)
        TL1     =       (256 - (FOSC/32/BAUD)); //设置波特率重装值
        TH1     =       (256 - (FOSC/32/BAUD));
        TR1     =       1;                      //定时器1开始工作
        ES      =       1;                      //使能串口中断
        EA      =       1;

        SendString("STC15F2K60S2\r\nUart Test !\r\n");
        while(1);
}

/*----------------------------
UART 中断服务程序
-----------------------------*/
void Uart() interrupt 4 using 1
{
        if (RI)
        {
                RI = 0;                         //清除RI位
                P0 = SBUF;                      //P0显示串口数据
                P22 = RB8;                      //P2.2显示校验位
        }
        if (TI)
        {
                TI = 0;                         //清除TI位
                busy = 0;                       //清忙标志
        }
}

/*----------------------------
发送串口数据
----------------------------*/
void SendData(BYTE dat)
{
        while (busy);                           //等待前面的数据发送完成
        ACC = dat;                              //获取校验位P (PSW.0)
        if (P)                          //根据P来设置校验位
        {
        #if (PARITYBIT == ODD_PARITY)
                TB8 = 0;                        //设置校验位为0
        #elif (PARITYBIT == EVEN_PARITY)
                TB8 = 1;                        //设置校验位为1
        #endif
        }
```

STC MCU Limited

```
        else
        {
        #if (PARITYBIT == ODD_PARITY)
                TB8 = 1;                                //设置校验位为1
        #elif (PARITYBIT == EVEN_PARITY)
                TB8 = 0;                                //设置校验位为0
        #endif
        }
        busy = 1;
        SBUF = ACC;                                     //写数据到UART数据寄存器
}

/*----------------------------
发送字符串
----------------------------*/
void SendString(char *s)
{
        while (*s)                                      //检测字符串结束标志
        {
                SendData(*s++);                         //发送当前字符
        }
}
```

2. 汇编程序：

```
/*------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器1用作串口1的波特率发生器举例---------------------------------------- */
/* --- 研发顾问QQ：  800003751-------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ----------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                               */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                               */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#define   NONE_PARITY     0                     //无校验
#define   ODD_PARITY      1                     //奇校验
#define   EVEN_PARITY     2                     //偶校验
#define   MARK_PARITY     3                     //标记校验
#define   SPACE_PARITY    4                     //空白校验

#define   PARITYBIT  EVEN_PARITY                //定义校验位

//---------------------------------------
```

STC MCU Limited

```
AUXR    EQU     08EH                    //辅助寄存器
BUSY    BIT     20H.0                   //忙标志位

//-----------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     0023H
        LJMP    UART_ISR
//-----------------------------------------
        ORG     0100H
MAIN:
        CLR     BUSY
        CLR     EA
        MOV     SP,     #3FH

#if (PARITYBIT == NONE_PARITY)
        MOV     SCON,   #50H            //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        MOV     SCON,   #0DAH           //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     SCON,   #0D2H           //9位可变波特率,校验位初始为0
#endif

//---------------------------------
        MOV     AUXR,   #40H            //定时器1为1T模式
        MOV     TMOD,   #20H            //定时器1为模式2(8位自动重载)
        MOV     TL1,    #0FBH           //设置波特率重装值(256-18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1                     //定时器1开始运行
        SETB    ES                      //使能串口中断
        SETB    EA

        MOV     DPTR,   #TESTSTR        //发送测试字符串
        LCALL   SENDSTRING

        SJMP    $
;-----------------------------------------
TESTSTR:
        DB  "STC15F2K60S2 Uart1 Test !",0DH,0AH,0

;/*---------------------------
;UART 中断服务程序
;---------------------------*/
UART_ISR:
        PUSH    ACC
        PUSH    PSW
        JNB     RI,     CHECKTI         //检测RI位
        CLR     RI                      //清除RI位
        MOV     P0,     SBUF            //P0显示串口数据
        MOV     C,      RB8
```

STC MCU Limited

```
        MOV     P2.2,   C                       //P2.2显示校验位
CHECKTI:
        JNB     TI,     ISR_EXIT                //检测TI位
        CLR     TI                              //清除TI位
        CLR     BUSY                            //清忙标志
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

;/*---------------------------
;发送串口数据
;---------------------------*/
SENDDATA:
        JB      BUSY,   $                       //等待前面的数据发送完成
        MOV     ACC,    A                       //获取校验位P (PSW.0)
        JNB     P,      EVEN1INACC              //根据P来设置校验位
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        CLR     TB8                             //设置校验位为0
#elif (PARITYBIT == EVEN_PARITY)
        SETB    TB8                             //设置校验位为1
#endif
        SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        SETB    TB8                             //设置校验位为1
#elif (PARITYBIT == EVEN_PARITY)
        CLR     TB8                             //设置校验位为0
#endif
PARITYBITOK:                                    //校验位设置完成
        SETB    BUSY
        MOV     SBUF,   A                       //写数据到UART数据寄存器
        RET

;/*---------------------------
;发送字符串
//---------------------------*/
SENDSTRING:
        CLR     A
        MOVC    A,      @A+DPTR                 //读取字符
        JZ      STRINGEND                       //检测字符串结束标志
        INC     DPTR                            //字符串地址+1
        LCALL   SENDDATA                        //发送当前字符
        SJMP    SENDSTRING
STRINGEND:
        RET
//-----------------------------------------
        END
```

STC MCU Limited

8.5 串行口2的相关寄存器

符号	描述	地址	位地址及符号 MSB LSB								复位值
S2CON	Serial 2 Control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000 0000B
S2BUF	Serial 2 Buffer	9BH									xxxx xxxxB
T2H	定时器2高8位，装入重装数	D6H									0000 0000B
T2L	定时器2低8位，装入重装数	D7H									0000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IE2	Interrupt Enable 2	AFH	-	-	-	-	-	-	ESPI	ES2	xxxx xx00B
IP2	Interrupt Priority 2 Low	B5H	-	-	-	-	-	-	PSPI	PS2	x000 0000B
P_SW2	外围设备功能切换控制寄存器	BAH	-	-	-	-	-	S4_S	S3_S	S2_S	xxxx x000B

1. 串行口2的控制寄存器S2CON

串行口2控制寄存器S2CON用于确定串行口2的工作方式和某些控制功能。其格式如下：

S2CON：串行口2控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	name	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0：指定串行口2的工作方式，如下表所示。

S2SM0	工作方式	功能说明	波特率
0	方式0	8位UART，波特率可变	(定时器T2的溢出率)/4
1	方式1	9位UART，波特率可变	(定时器T2的溢出率)/4
当AUXR.2/T2x12=1时，定时器T2的溢出率 = SYSclk / (65536 - [RL_TH2,RL_TL2]) 当AUXR.2/T2x12=0时，定时器T2的溢出率 = SYSclk / 12 / (65536 - [RL_TH2,RL_TL2]) 式中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。			

B6:保留，该位复位后为0.

S2SM2：允许方式1多机通信控制位。

在方式1时，如果S2SM2位为1且S2REN位为1，则接收机处于地址帧筛选状态。此时可以利用接收到的第9位(即S2RB8)来筛选地址帧：若S2RB8=1，说明该帧是地址帧，地址信息可以进入S2BUF，并使S2RI为1，进而在中断服务程序中再进行地址号比较；若S2RB8=0，说明该帧不是地址帧，应丢掉且保持S2RI=0。在方式1中，如果S2SM2位为0且S2REN位为1，接收收机处于地址帧筛选被禁止状态。不论收到的S2RB8为0或1，均可使接收到的信息进入S2BUF, 并使S2RI=1, 此时S2RB8通常为校验位.

方式0是非多机通信方式，在这种方式时，要设置S2SM2 应为0。

STC MCU Limited

S2REN：允许/禁止串行口2接收控制位。由软件置位S2REN，即S2REN=1为允许串行接收状态，可启动串行接收器RxD2，开始接收信息。软件复位S2REN，即S2REN=0，则禁止接收。

S2TB8：在方式1，S2TB8为要发送的第9位数据，按需要由软件置位或清0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式0中，该位不用.

S2RB8：在方式1，S2RB8是接收到的第9位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式0中不用S2RB8(置S2SM2=0, S2RB8是接收到的停止位)。

S2TI：发送中断请求标志位。在停止位开始发送时由S2TI内部硬件置位，即S2TI=1, 响应中断后S2TI必须用软件清零。

S2RI：接收中断请求标志位。在串行接收到停止位的中间时刻S2RI由内部硬件置位，即S2RI=1, 向CPU发中断申请，响应中断后S2RI必须由软件清零。

S2CON的所有位可通过整机复位信号复位为全“0”。S2CON的字节地址为9AH，不可位寻址。串行通信的中断请求：当一帧发送完成，内部硬件自动置位S2TI，即S2TI=1，请求中断处理；当接收完一帧信息时，内部硬件自动置位S2RI，即S2RI=1，请求中断处理。由于S2TI和S2RI以**“或逻辑”关系向主机请求中断，所以主机响应中断时事先并不知道是**S2TI还是S2RI请求的中断，必须在中断服务程序中查询S2TI和S2RI进行判别，然后分别处理。因此，两个中断请求标志位均不能由硬件自动置位，必须通过软件清0，否则将出现一次请求多次响应的错误。

STC MCU Limited

2. 串行口2的数据缓冲寄存器S2BUF

STC15系列单片机的串行口2数据缓冲寄存器(S2BUF)的地址是9BH，实际是2个缓冲器，写S2BUF的操作完成待发送数据的加载，读S2BUF的操作可获得已接收到的数据。两个操作分别对应两个不同的寄存器，1个是只写寄存器，1个是只读寄存器。

串行通道内设有数据寄存器。在所有的串行通信方式中，在写入S2BUF信号（MOV S2BUF,A）的控制下，把数据装入相同的9位移位寄存器，前面8位为数据字节，其最低位为移位寄存器的输出位。根据不同的工作方式会自动将“1”或S2TB8的值装入移位寄存器的第9位, 并进行发送.

串行通道的接收寄存器是一个输入移位寄存器。在方式0和方式1时均为9位。当一帧接收完毕，移位寄存器中的数据字节装入串行数据缓冲器S2BUF中, 其第9位则装入S2CON寄存器中的S2RB8位。如果由于S2SM2使得已接收到的数据无效时, S2RB8和S2BUF中内容不变.

由于接收通道内设有输入移位寄存器和S2BUF缓冲器，从而能使一帧接收完将数据由移位寄存器装入S2BUF后，可立即开始接收下一帧信息，主机应在该帧接收结束前从S2BUF缓冲器中将数据取走，否则前一帧数据将丢失。S2BUF以并行方式送往内部数据总线。

3. 串行口2只能选择定时器2作为其波特率发生器——定时器2的寄存器T2H, T2L

定时器2寄存器T2H(地址为D6H，复位值为00H)及寄存器T2L(地址为D7H，复位值为00H)用于保存重装时间常数。

注意：对于STC15系列单片机，串口2永远是使用定时器2作为波特率发生器,串口2不能够选择其他定时器作其波特率发生器；串口1默认选择定时器2作为其波特率发生器，也可以选择定时器1作为其波特率发生器；串口3默认选择定时器2作为其波特率发生器，也可以选择定时器3作为其波特率发生器；串口4默认选择定时器2作为其波特率发生器，也可以选择定时器4作为其波特率发生器。

4. 定时器2的控制位——TR2、T2_C/$\overline{T}$、T2x12

AUXR：辅助寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2

T2R：定时器2运行控制位

0，不允许定时器2运行；

1，允许定时器2运行

T2_C/$\overline{T}$：控制定时器2用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T2/P3.1的外部脉冲进行计数)

T2x12：定时器2速度控制位

0，定时器2是传统8051速度，12分频；

1，定时器2的速度是传统8051的12倍，不分频

如果串口1或串口2用T2作为波特率发生器，则由T2x12决定串口1或串口2是12T还是1T.

对于STC15系列单片机，串口2只能使用定时器2作为波特率发生器，不能够选择其他定时器作为其波特率发生器；而串口1默认选择定时器2作为其波特率发生器，也可以选择定时器1作为其波特率发生器；串口3默认选择定时器2作为其波特率发生器，也可以选择定时器3作为其波特率发生器；串口4默认选择定时器2作为其波特率发生器，也可以选择定时器4作为其波特率发生器。

STC MCU Limited

5. 与串行口2中断相关的寄存器

串行口2中断允许位ES2位于中断允许寄存器IE2中，中断允许寄存器的格式如下：

IE2：中断允许寄存器2（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	-	-	-	-	-	ESPI	ES2

ES2：串行口2中断允许位，ES2=1，允许串行口2中断，ES2=0，禁止串行口2中断。

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的总中断允许控制位，EA=1，CPU开放中断，EA=0，CPU屏蔽所有的中断申请。
EA的作用是使中断允许形成多级控制。即各中断源首先受EA控制;其次还受各中断源自己的中断允许控制位控制。

串行口2中断优先级控制位PS2位位于中断优先级控制寄存器IP中，中断优先级控制寄存器的格式如下：

IP2：中断优先级控制寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP2	B5H	name	-	-	-	-	-	-	PSPI	PS2

PS2: 串行口2中断优先级控制位。
当PS2=0时，串行口2中断为最低优先级中断(优先级0)
当PS2=1时，串行口2中断为最高优先级中断(优先级1)

6. 串行口2在2组管脚之间切换的控制位——S2_S/P_SW2.0

通过设置寄存器P_SW2中的S2_S位，可以将串口2在2组管脚之间任意切换，P_SW2寄存器的格式如下：

P_SW2: 外围设备功能切换控制寄存器2（不可位寻址）

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
P_SW2	BAH	外围设备功能切换控制寄存器2						S4_S	S3_S	S2_S	xxxx,x000

串口2/S2可在2个地方切换，由 S2_S 控制位来选择	
S2_S	S2可在P1/P4之间来回切换
0	串口2/S2在[P1.0/RxD2,P1.1/TxD2]
1	串口2/S2在[P4.6/RxD2_2,P4.7/TxD2_2]

串口3/S3可在2个地方切换，由 S3_S 控制位来选择	
S3_S	S3可在P0/P5之间来回切换
0	串口3/S3在[P0.0/RxD3,P0.1/TxD3]
1	串口3/S3在[P5.0/RxD3_2,P5.1/TxD3_2]

串口4/S4可在2个地方切换，由 S4_S 控制位来选择	
S4_S	S4可在P0/P5之间来回切换
0	串口4/S4在[P0.2/RxD4,P0.3/TxD4]
1	串口4/S4在[P5.2/RxD4_2,P5.3/TxD4_2]

STC MCU Limited

8.6 串行口2工作模式

——串口2固定使用定时器T2作波特率发生器

——串口1/3/4和串口2的波特率相同时,串口1/3/4和串口2可共享T2作波特率发生器

STC15W4K60S4系列单片机的串行口2有两种工作模式，可通过软件编程对S2CON中的S2SM0的设置进行选择。其中模式0和模式1都为异步通信，每个发送和接收的字符都带有1个启动位和1个停止位。

8.6.1 串行口2的工作模式0----8位UART，波特率可变

10位数据通过RxD2/P1.0(RxD2_2/P4.6)接收，通过TxD2/P1.1(TxD2_2/P4.7)发送。一帧数据包含一个起始位(0)，8个数据位和一个停止位(1)。接收时，停止位进入特殊功能寄存器S2CON的S2RB8位。波特率由定时器T2的溢出率决定。

串口2在模式0的波特率 = 定时器T2的溢出率/4

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率=SYSclk / (65536 - [RL_TH2, RL_TL2])；

即此时，串行口2的波特率=SYSclk / (65536 - [RL_TH2, RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2, RL_TL2])；

即此时，串行口2的波特率=SYSclk / 12 / (65536 - [RL_TH2, RL_TL2]) / 4

上式中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。

8.6.2 串行口2的工作模式1----9位UART，波特率可变

11位数据通过RxD2/P1.0(RxD2_2/P4.6)发送，通过TxD2/P1.1(TxD2_2/P4.7)接收。一帧数据包含一个起始位(0)，8个数据位，一个可编程的第9位，和一个停止位(1)。发送时，第9位数据位来自特殊功能寄存器S2CON的S2TB8位. 接收时，第9位进入特殊功能寄存器S2CON的S2RB8位。

串口2在模式1的波特率= T2 定时器2的溢出率/4

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率=SYSclk / (65536 - [RL_TH2, RL_TL2])；

即此时，串行口2的波特率=SYSclk / (65536 - [RL_TH2, RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2, RL_TL2])；

即此时，串行口2的波特率=SYSclk / 12 / (65536 - [RL_TH2, RL_TL2]) / 4

上式中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。

可见，模式1和模式0一样，其波特率可通过软件对定时器2的设置进行波特率的选择，是可变的。

STC-MCU Limited

说明：当串口1、串口3及串口4和串口2的波特率相同时，串口1、串口3及串口4和串口2可以共享定时器T2作波特率发生器，此时建议串口1、串口3及串口4都选择定时器T2作为波特率发生器；

当串口1、串口3及串口4和串口2的波特率不同时，串口1、串口3及串口4和串口2不可以共享定时器T2作波特率发生器，这是才建议串口1选择定时器T1作波特率发生器，串口3选择定时器T3作波特率发生器，串口4选择定时器T4作波特率发生器。

用户在程序中如何具体使用串口2

1. 设置串口2的工作模式，S2CON寄存器中的S2SM0决定了串口2的2种工作模式
2. 设置串口2的波特率相应的寄存器：

   定时器2寄存器T2H / T2L
3. 启动定时器2，让T2R位为1，定时器2就立即开始计数。
4. 设置AUXR. 2/T2x12, 确定定时器2的速度
5. 设置串口2的中断优先级，及打开中断相应的控制位是:

   PS2, PS2H, ES2, EA
6. 如要串口2接收，将S2REN置1 即可

   如要串口2发送，将数据送入S2BUF即可，

   接收完成标志S2RI, 发送完成标志S2TI, 要由软件清0。

STC MCU Limited

8.7 串行口2的测试程序(C和汇编)

——使用定时器2作串口2的波特率发生器

1. C程序:

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器2用作串口2的波特率发生器举例---------------------- */
/* --- 研发顾问QQ：800003751----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----              */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----              */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define  FOSC    18432000L                      //系统频率
#define  BAUD    115200                         //串口波特率
#define  TM      (65536 - (FOSC/4/BAUD))

#define  NONE_PARITY        0                   //无校验
#define  ODD_PARITY         1                   //奇校验
#define  EVEN_PARITY        2                   //偶校验
#define  MARK_PARITY        3                   //标记校验
#define  SPACE_PARITY       4                   //空白校验

#define PARITYBIT EVEN_PARITY                   //定义校验位

sfr      AUXR    =    0x8e;                     //辅助寄存器
sfr      S2CON   =    0x9a;                     //UART2 控制寄存器
sfr      S2BUF   =    0x9b;                     //UART2 数据寄存器
sfr      T2H     =    0xd6;                     //定时器2高8位
sfr      T2L     =    0xd7;                     //定时器2低8位
sfr      IE2     =    0xaf;                     //中断控制寄存器2

#define  S2RI    0x01                           //S2CON.0
#define  S2TI    0x02                           //S2CON.1
```

```
#define   S2RB8   0x04                    //S2CON.2
#define   S2TB8   0x08                    //S2CON.3

bit       busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
        S2CON = 0x50;                     //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
         S2CON = 0xda;                    //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        S2CON = 0xd2;                     //9位可变波特率,校验位初始为0
#endif

        T2L = TM;                         //设置波特率重装值
        T2H = TM>>8;
        AUXR = 0x14;                      //T2为1T模式, 并启动定时器2
        IE2 = 0x01;                       //使能串口2中断
        EA = 1;

        SendString("STC15F2K60S2\r\nUart2 Test !\r\n");
        while(1);
}

/*----------------------------
UART2 中断服务程序
----------------------------*/
void Uart2() interrupt 8 using 1
{
        if (S2CON & S2RI)
        {
                S2CON &= ~S2RI;           //清除S2RI位
                P0 = S2BUF;               //P0显示串口数据
                P2 = (S2CON & S2RB8);      //P2.2显示校验位
         }
        if (S2CON & S2TI)
        {
                S2CON &= ~S2TI;           //清除S2TI位
                busy = 0;                 //清忙标志
        }
}

/*----------------------------
```

STC MCU Limited

```
发送串口数据
------------------------------*/
void SendData(BYTE dat)
{
        while (busy);                           //等待前面的数据发送完成
        ACC = dat;                              //获取校验位P (PSW.0)
        if (P)                                  //根据P来设置校验位
        {
        #if (PARITYBIT == ODD_PARITY)
                S2CON &= ~S2TB8;                //设置校验位为0
        #elif (PARITYBIT == EVEN_PARITY)
                S2CON |= S2TB8;                 //设置校验位为1
        #endif
        }
        else
        {
        #if (PARITYBIT == ODD_PARITY)
                S2CON |= S2TB8;                 //设置校验位为1
        #elif (PARITYBIT == EVEN_PARITY)
                S2CON &= ~S2TB8;                //设置校验位为0
        #endif
        }
        busy = 1;
        S2BUF = ACC;                            //写数据到UART2数据寄存器
}

/*----------------------------
发送字符串
----------------------------*/
void SendString(char *s)
{
        while (*s)                              //检测字符串结束标志
        {
                SendData(*s++);                 //发送当前字符
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 定时器2用作串口2的波特率发生器举例---------------------------------------- */
/* --- 研发顾问QQ：  800003751----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#define   NONE_PARITY       0       //无校验
#define   ODD_PARITY        1       //奇校验
#define   EVEN_PARITY       2       //偶校验
#define   MARK_PARITY       3       //标记校验
#define   SPACE_PARITY      4       //空白校验

#define PARITYBIT EVEN_PARITY       //定义校验位

//---------------------------------------

AUXR    EQU     08EH                //辅助寄存器
S2CON   EQU     09AH                //UART2 控制寄存器
S2BUF   EQU     09BH                //UART2 数据寄存器
T2H     DATA    0D6H                //定时器2高8位
T2L     DATA    0D7H                //定时器2低8位
IE2     EQU     0AFH                //中断控制寄存器2

S2RI    EQU     01H                 //S2CON.0
S2TI    EQU     02H                 //S2CON.1
S2RB8   EQU     04H                 //S2CON.2
S2TB8   EQU     08H                 //S2CON.3
//---------------------------------------
BUSY    BIT     20H.0               //忙标志位
//---------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     0043H
        LJMP    UART2_ISR
//---------------------------------------
        ORG     0100H
```

STC MCU Limited

```
MAIN:
        CLR     BUSY
        CLR     EA
        MOV     SP,     #3FH
#if (PARITYBIT == NONE_PARITY)
        MOV     S2CON,  #50H                    //8位可变波特率
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        MOV     S2CON,  #0DAH                   //9位可变波特率,校验位初始为1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     S2CON,  #0D2H                   //9位可变波特率,校验位初始为0
#endif

//---------------------------------
        MOV     T2L,    #0D8H                   //设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H                    //T2为1T模式, 并启动定时器2
        ORL     IE2,    #01H                    //使能串口2中断
        SETB    EA

        MOV     DPTR,   #TESTSTR                //发送测试字符串
        LCALL   SENDSTRING

        SJMP    $
;----------------------------------------
TESTSTR:
        DB  "STC15F2K60S2 Uart2 Test !",0DH,0AH,0

;/*---------------------------
;UART2 中断服务程序
;---------------------------*/
UART2_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     A,      S2CON                   ;读取UART2控制寄存器
        JNB     ACC.0,  CHECKTI                 ;检测S2RI位
        ANL     S2CON,  #NOT S2RI               ;清除S2RI位
         MOV    P0,     S2BUF                   ;P0显示串口数据
        ANL     A,      #S2RB8                  ;
        MOV     P2,     A                       ;P2.2显示校验位
CHECKTI:                        ;
        MOV     A,      S2CON                   ;读取UART2控制寄存器
        JNB     ACC.1,  ISR_EXIT                ;检测S2TI位
        ANL     S2CON,  #NOT S2TI               ;清除S2TI位
        CLR     BUSY                            ;清忙标志
ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI
```

STC MCU Limited

```
;/*----------------------------
;发送串口数据
;----------------------------*/
SENDDATA:
        JB      BUSY,   $               //等待前面的数据发送完成
        MOV     ACC,    A               //获取校验位P (PSW.0)
        JNB     P,      EVEN1INACC      //根据P来设置校验位
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        ANL     S2CON,  #NOT S2TB8      //设置校验位为0
#elif (PARITYBIT == EVEN_PARITY)
        ORL     S2CON,  #S2TB8          //设置校验位为1
#endif
        SJMP    PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        ORL     S2CON,  #S2TB8          //设置校验位为1
#elif (PARITYBIT == EVEN_PARITY)
        ANL     S2CON,  #NOT S2TB8      //设置校验位为0
#endif
PARITYBITOK:                            //校验位设置完成
        SETB    BUSY
        MOV     S2BUF,  A               //写数据到UART2数据寄存器
        RET

;/*----------------------------
;发送字符串
//----------------------------*/
SENDSTRING:
        CLR     A
        MOVC    A,      @A+DPTR         //读取字符
        JZ      STRINGEND               //检测字符串结束标志
        INC     DPTR                    //字符串地址+1
        LCALL   SENDDATA                //发送当前字符
        SJMP    SENDSTRING
STRINGEND:
        RET
//-----------------------------------------
        END
```

STC MCU Limited

8.8 串行口3的相关寄存器

符号	描述	地址	位地址及符号 MSB LSB								复位值
S3CON	串口3控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口3数据缓冲器	ADH									xxxx,xxxx
T2H	定时器2高8位，装入重装数	D6H									0000 0000B
T2L	定时器2低8位，装入重装数	D7H									0000 0000B
AUXR	辅助寄存器	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
T3H	定时器3高8位寄存器	D4H									0000 0000B
T3L	定时器3低8位寄存器	D5H									0000 0000B
T4T3M	T4和T3的控制寄存器	D1H	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO	0000 0000B
IE2	中断允许寄存器	AFH			ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
P_SW2	外围设备功能切换控制寄存器	BAH	-	-	-	-	-	S4_S	S3_S	S2_S	xxxx x000B

1. 串行口3的控制寄存器S3CON

串行口3控制寄存器S3CON用于确定串行口3的工作方式和某些控制功能。其格式如下：

S3CON：串行口3控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	name	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0：指定串行口3的工作方式，如下表所示。

S3SM0	工作方式	功能说明	波特率
0	方式0	8位UART，波特率可变	(定时器T2的溢出率)/4 或 (定时器T3的溢出率)/4
1	方式1	9位UART，波特率可变	(定时器T2的溢出率)/4 或 (定时器T3的溢出率)/4

当AUXR.2/T2x12=1时，定时器T2的溢出率 = SYSclk / (65536 - [RL_TH2,RL_TL2])
当AUXR.2/T2x12=0时，定时器T2的溢出率 = SYSclk / 12 / (65536 - [RL_TH2,RL_TL2])
式中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。

当T4T3M.1/T3x12=1时，定时器T3的溢出率 = SYSclk / (65536 - [RL_TH3, RL_TL3])
当T4T3M.1/T3x12=0时，定时器T3的溢出率 = SYSclk / 12 / (65536 - [RL_TH3, RL_TL3])
式中RL_TH3是T3H的重装载寄存器，RL_TL3是T3L的重装载寄存器。

S3ST3：串口3(UART3)选择定时器3作波特率发生器的控制位。

0，**串行口3选择定时器2作为其波特率发生器；**

1，**串行口3选择定时器3作为其波特率发生器**

S3SM2：允许方式1多机通信控制位。

在方式1时，如果S3SM2位为1且S3REN位为1，则接收机处于地址帧筛选状态。此时可以利用接收到的第9位(即S3RB8)来筛选地址帧：若S3RB8=1，说明该帧是地址帧，地址信息可以进入S3BUF，并使S3RI为1，进而在中断服务程序中再进行地址号比较；若S3RB8=0，说明该帧不是地址帧，应丢掉且保持S3RI=0。在方式1中，如果S3SM2位为0且S3REN位为1，接收收机处于地址帧筛选被禁止状态。不论收到的S3RB8为0或1，均可使接收到的信息进入S3BUF,并使S3RI=1,此时S3RB8通常为校验位.
方式0是非多机通信方式，在这种方式时，要设置S3SM2 应为0。

S3REN：允许/禁止串行口3接收控制位。由软件置位S3REN，即S3REN=1为允许串行接收状态，可启动串行接收器RxD3，开始接收信息。软件复位S3REN，即S3REN=0，则禁止接收。

S3TB8：在方式1，S3TB8为要发送的第9位数据，按需要由软件置位或清0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式0中，该位不用.

S3RB8：在方式1，S3RB8是接收到的第9位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式0中不用S3RB8(置S3SM2=0, S3RB8是接收到的停止位)。

S3TI：发送中断请求标志位。在停止位开始发送时由S3TI内部硬件置位，即S3TI=1,响应中断后S3TI必须用软件清零。

S3RI：接收中断请求标志位。在串行接收到停止位的中间时刻S3RI由内部硬件置位，即S3RI=1,向CPU发中断申请，响应中断后S3RI必须由软件清零。

S3CON的所有位可通过整机复位信号复位为全“0”。S3CON的字节地址为ACH，不可位寻址。串行通信的中断请求：当一帧发送完成，内部硬件自动置位S3TI，即S3TI=1，请求中断处理；当接收完一帧信息时，内部硬件自动置位S3RI，即S3RI=1，请求中断处理。由于S3TI和S3RI以**“或逻辑”关系向主机请求中断，所以主机响应中断时事先并不知道是**S3TI还是S3RI请求的中断，必须在中断服务程序中查询S3TI和S3RI进行判别，然后分别处理。因此，两个中断请求标志位均不能由硬件自动置位，必须通过软件清0，否则将出现一次请求多次响应的错误。

2. 串行口3的数据缓冲寄存器S3BUF

STC15W4K60S4系列单片机的串行口3数据缓冲寄存器(S3BUF)的地址是ADH，实际是2个缓冲器，写S3BUF的操作完成待发送数据的加载，读S3BUF的操作可获得已接收到的数据。两个操作分别对应两个不同的寄存器，1个是只写寄存器，1个是只读寄存器。

串行通道内设有数据寄存器。在所有的串行通信方式中，在写入S3BUF信号（MOV S3BUF,A）的控制下，把数据装入相同的9位移位寄存器，前面8位为数据字节，其最低位为移位寄存器的输出位。根据不同的工作方式会自动将“1”或S3TB8的值装入移位寄存器的第9位, 并进行发送.

串行通道的接收寄存器是一个输入移位寄存器。在方式0和方式1时均为9位。当一帧接收完毕，移位寄存器中的数据字节装入串行数据缓冲器S3BUF中, 其第9位则装入S3CON寄存器中的S3RB8位。如果由于S3SM2使得已接收到的数据无效时, S3RB8和S3BUF中内容不变.

由于接收通道内设有输入移位寄存器和S3BUF缓冲器，从而能使一帧接收完将数据由移位寄存器装入S3BUF后，可立即开始接收下一帧信息，主机应在该帧接收结束前从S3BUF缓冲器中将数据取走，否则前一帧数据将丢失。S3BUF以并行方式送往内部数据总线。

3. 串行口3既能选择定时器2作为其波特率发生器，也能选择定时器3作为其波特率发生器——定时器2的寄存器T2H, T2L和定时器3的寄存器T3H, T3L

定时器2寄存器T2H(地址为D6H，复位值为00H)及寄存器T2L(地址为D7H，复位值为00H)用于保存重装时间常数。

定时器3寄存器T3H(地址为D4H，复位值为00H)及寄存器T3L(地址为D5H，复位值为00H)用于保存重装时间常数。

注意：有串口2的单片机，串口2永远是使用定时器2作为波特率发生器, 串口2不能够选择定时器1 做波特率发生器，串口1可以选择定时器1做波特率发生器，也可以选择定时器2作为波特率发生器。而串口3可以选择定时器2做波特率发生器，也可以选择定时器3作为波特率发生器。同样串口4可以选择定时器2做波特率发生器，也可以选择定时器4作为波特率发生器。

STC MCU Limited

4. 定时器2的控制位——TR2、T2_C/$\overline{T}$、T2x12

AUXR：辅助寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2

T2R：定时器2运行控制位

0，不允许定时器2运行；

1，允许定时器2运行

T2_C/$\overline{T}$：控制定时器2用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T2/P3.1的外部脉冲进行计数)

T2x12：定时器2速度控制位

0，定时器2是传统8051速度，12分频；

1，定时器2的速度是传统8051的12倍，不分频

如果串口1或串口2用T2作为波特率发生器，则由T2x12决定串口1或串口2是12T还是1T.

5. 定时器3的控制位——TR3、T3_C/$\overline{T}$、T3x12

T4T3M(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO

B3 - T3R：定时器3运行控制位。

0，不允许定时器3运行；

1，允许定时器3运行。

B2 - T3_C/$\overline{T}$：控制定时器3用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T3/P0.5的外部脉冲进行计数)

B1 - T3x12：定时器3速度控制位。

0，定时器3速度是8051单片机定时器的速度，即12分频；

1，定时器3速度是8051单片机定时器速度的12倍，即不分频。

STC MCU Limited

6. 与串行口3中断相关的寄存器IE2

串行口3中断允许位ES3位于中断允许寄存器IE2中，中断允许寄存器的格式如下：

IE2：中断允许寄存器2（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4：定时器4的中断允许位。

1，允许定时器4产生中断；

0，禁止定时器4产生中断。

ET3：定时器3的中断允许位。

1，允许定时器3产生中断；

0，禁止定时器3产生中断。

ES4：串行口4中断允许位。

1，允许串行口4中断；

0，禁止串行口4中断

ES3：串行口3中断允许位。

1，允许串行口3中断；

0，禁止串行口3中断。

ET2：定时器2的中断允许位。

1，允许定时器2产生中断；

0，禁止定时器2产生中断。

ESPI：SPI中断允许位。

1，允许SPI中断；

0，禁止SPI中断。

ES2：串行口2中断允许位。

1，允许串行口2中断；

0，禁止串行口2中断。

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的总中断允许控制位

1，CPU开放中断，

0，CPU屏蔽所有的中断申请。

EA的作用是使中断允许形成多级控制。即各中断源首先受EA控制;其次还受各中断源自己的中断允许控制位控制。

STC MCU Limited

7. 串行口3在2组管脚之间切换的控制位——S3_S/P_SW2.1

通过设置寄存器P_SW2中的S3_S位，可以将串口3在2组管脚之间任意切换，P_SW2寄存器的格式如下：

P_SW2：外围设备切换控制寄存器2（不可位寻址）

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
P_SW2	BAH	外围设备功能切换控制寄存器2						S4_S	S3_S	S2_S	xxxx,x000

串口2/S2可在2个地方切换，由 S2_S 控制位来选择	
S2_S	S2可在P1/P4之间来回切换
0	串口2/S2在[P1.0/RxD2,P1.1/TxD2]
1	串口2/S2在[P4.6/RxD2_2,P4.7/TxD2_2]

串口3/S3可在2个地方切换，由 S3_S 控制位来选择	
S3_S	S3可在P0/P5之间来回切换
0	串口3/S3在[P0.0/RxD3,P0.1/TxD3]
1	串口3/S3在[P5.0/RxD3_2,P5.1/TxD3_2]

串口4/S4可在2个地方切换，由 S4_S 控制位来选择	
S4_S	S4可在P0/P5之间来回切换
0	串口4/S4在[P0.2/RxD4,P0.3/TxD4]
1	串口4/S4在[P5.2/RxD4_2,P5.3/TxD4_2]

STC MCU Limited

8.9 串行口3工作模式

——串口3和串口2的波特率相同时,串口3和串口2可共享T2作波特率发生器

STC15W4K60S4系列单片机的串行口3有两种工作模式，可通过软件编程对S3CON中的S3SM0的设置进行选择。其中模式0和模式1都为异步通信，每个发送和接收的字符都带有1个启动位和1个停止位。

8.9.1 串行口3的工作模式0----8位UART，波特率可变

10位数据通过RxD3/P0.0(RxD3/P5.0)接收，通过TxD3/P0.1(TxD3/P5.1)发送。一帧数据包含一个起始位(0)，8个数据位和一个停止位(1)。接收时，停止位进入特殊功能寄存器S3CON的S3RB8位。串行口3既可以选择定时器2作其波特率发生器，也可以选择定时器3作其波特率发生器。所以串行口3的波特率由定时器T2的溢出率或定时器T3的溢出率决定。

当串行口3选择定时器T2作为其波特率发生器(即S3ST3/S3SCON.1=0)时：

串口3在模式0的波特率 = 定时器T2的溢出率/4

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率=SYSclk / (65536 - [RL_TH2, RL_TL2])；

即此时，串行口3的波特率=SYSclk / (65536 - [RL_TH2, RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2, RL_TL2])；

即此时，串行口3的波特率=SYSclk / 12 / (65536 - [RL_TH2, RL_TL2]) / 4

上式中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。

当串行口3选择定时器T3作为其波特率发生器(即S3ST3/S3SCON.1=1)时：

串口3波特率在模式0 = 定时器T3的溢出率/4

当T3工作在1T模式(T4T3M.1/T3x12=1)时，定时器3的溢出率=SYSclk / (65536 - [RL_TH3, RL_TL3])；

即此时，串行口3的波特率=SYSclk / (65536 - [RL_TH3, RL_TL3]) / 4

当T3工作在12T模式(T4T3M.1/T3x12=0)时，定时器3的溢出率=SYSclk /12/(65536 - [RL_TH3, RL_TL3])；

即此时，串行口3的波特率=SYSclk / 12 / (65536 - [RL_TH3, RL_TL3]) / 4

上式中RL_TH3是T3H的重装载寄存器，RL_TL3是T3L的重装载寄存器。

说明：当串口3和串口2的波特率相同时，串口3和串口2可以共享波特率发生器，此时建议用户选择定时器T2作为串口3的波特率发生器；当串口3和串口2的波特率不同时，才建议选择定时器T3作为串口3的波特率发生器(因串口2固定使用定时器T2作波特率发生器)。

STC-MCU Limited

8.9.2 串行口3的工作模式1----9位UART，波特率可变

11位数据通过TxD3/P0.1(TxD3/P5.1)发送，通过RxD3/P0.0(RxD3/P5.0)接收。一帧数据包含一个起始位(0)，8个数据位，一个可编程的第9位，和一个停止位(1)。发送时，第9位数据位来自特殊功能寄存器S3CON的S3TB8位. 接收时，第9位进入特殊功能寄存器S3CON的S3RB8位。串行口3既可以选择定时器2作其波特率发生器，也可以选择定时器3作其波特率发生器。所以串行口3的波特率由定时器T2的溢出率或定时器T3的溢出率决定。

当串行口3选择定时器T2作为其波特率发生器(即S3ST3/S3SCON. 1=0)时：

串口3在模式1的波特率 = 定时器T2的溢出率/4

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率=SYSclk / (65536 - [RL_TH2, RL_TL2])；

即此时，串行口3的波特率=SYSclk / (65536 - [RL_TH2, RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2, RL_TL2])；

即此时，串行口3的波特率=SYSclk / 12 / (65536 - [RL_TH2, RL_TL2]) / 4

上式中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。

当串行口3选择定时器T3作为其波特率发生器(即S3ST3/S3SCON. 1=1)时：

串口3波特率在模式1= 定时器T3的溢出率/4

当T3工作在1T模式(T4T3M.1/T3x12=1)时，定时器3的溢出率=SYSclk / (65536 - [RL_TH3, RL_TL3])；

即此时，串行口3的波特率=SYSclk / (65536 - [RL_TH3, RL_TL3]) / 4

当T3工作在12T模式(T4T3M.1/T3x12=0)时，定时器3的溢出率=SYSclk /12/(65536 - [RL_TH3, RL_TL3])；

即此时，串行口3的波特率=SYSclk / 12 / (65536 - [RL_TH3, RL_TL3]) / 4

上式中RL_TH3是T3H的重装载寄存器，RL_TL3是T3L的重装载寄存器。

可见，模式1和模式0一样，其波特率可通过软件对定时器2或定时器3的设置进行波特率的选择，是可变的。

说明：当串口3和串口2的波特率相同时，串口3和串口2可以共享波特率发生器，此时建议用户选择定时器T2作为串口3的波特率发生器；当串口3和串口2的波特率不同时，才建议选择定时器T3作为串口3的波特率发生器(因串口2固定使用定时器T2作波特率发生器)。

用户在程序中如何具体使用串口3

1. 设置串口3的工作模式，S3CON寄存器中的S3SM0决定了串口3的2种工作模式
2. 设置串口3的波特率相应的寄存器：

   定时器3寄存器T3H / T3L
3. 启动定时器3，让T3R位为1，定时器3就立即开始计数。
4. 设置T4T3M.1/T3x12, 确定定时器3的速度
5. 打开串口3中断，设置相应的控制位是：

   ES3, EA
6. 如要串口3接收，将S3REN置1 即可

   如要串口3发送，将数据送入S3BUF即可，

   接收完成标志S3RI, 发送完成标志S3TI, 要由软件清0。

STC MCU Limited

8.10 串行口4的相关寄存器

符号	描述	地址	位地址及符号 MSB LSB	复位值
S4CON	串口4控制寄存器	84H	S4SM0 | S4ST4 | S4SM2 | S4REN | S4TB8 | S4RB8 | S4TI | S4RI	0000,0000
S4BUF	串口4数据缓冲器	85H		xxxx,xxxx
T2H	定时器2高8位，装入重装数	D6H		0000 0000B
T2L	定时器2低8位，装入重装数	D7H		0000 0000B
AUXR	辅助寄存器	8EH	T0x12 | T1x12 | UART_M0x6 | T2R | T2_C/$\overline{T}$ | T2x12 | EXTRAM | S1ST2	0000 0001B
T4H	定时器4高8位寄存器	D2H		0000 0000B
T4L	定时器4低8位寄存器	D3H		0000 0000B
T4T3M	T4和T3的控制寄存器	D1H	T4R | T4_C/$\overline{T}$ | T4x12 | T4CLKO | T3R | T3_C/$\overline{T}$ | T3x12 | T3CLKO	0000 0000B
IE2	中断允许寄存器	AFH	| ET4 | ET3 | ES4 | ES3 | ET2 | ESPI | ES2	x000 0000B
P_SW2	外围设备功能切换控制寄存器	BAH	- | - | - | - | S4_S | S3_S | S2_S	xxxx x000B

STC MCU Limited

1. 串行口4的控制寄存器S4CON

串行口4控制寄存器S4CON用于确定串行口4的工作方式和某些控制功能。其格式如下：

S4CON：串行口4控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	name	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0：指定串行口4的工作方式，如下表所示。

S4SM0	工作方式	功能说明	波特率
0	方式0	8位UART，波特率可变	(定时器T4的溢出率)/4
1	方式1	9位UART，波特率可变	(定时器T4的溢出率)/4
当T4T3M.5/T4x12=1时，定时器T4的溢出率 = SYSclk / (65536 - [RL_TH4, RL_TL4]) 当T4T3M.5/T4x12=0时，定时器T4的溢出率 = SYSclk / 12 / (65536 - [RL_TH4, RL_TL4]) 式中RL_TH4是T4H的重装载寄存器，RL_TL4是T4L的重装载寄存器。			

S4ST4：串口4(UART4)选择定时器4作波特率发生器的控制位。

0，**串行口4选择定时器2作为其波特率发生器；**

1，**串行口4选择定时器4作为其波特率发生器**

S4SM2：允许方式1多机通信控制位。

在方式1时，如果S4SM2位为1且S4REN位为1，则接收机处于地址帧筛选状态。此时可以利用接收到的第9位(即S4RB8)来筛选地址帧：若S4RB8=1，说明该帧是地址帧，地址信息可以进入S4BUF，并使S4RI为1，进而在中断服务程序中再进行地址号比较；若S4RB8=0，说明该帧不是地址帧，应丢掉且保持S4RI=0。在方式1中，如果S4SM2位为0且S4REN位为1，接收收机处于地址帧筛选被禁止状态。不论收到的S4RB8为0或1，均可使接收到的信息进入S4BUF,并使S4RI=1,此时S4RB8通常为校验位.
方式0是非多机通信方式，在这种方式时，要设置S4SM2 应为0。

S4REN：允许/禁止串行口4接收控制位。由软件置位S4REN，即S4REN=1为允许串行接收状态，可启动串行接收器RxD4，开始接收信息。软件复位S4REN，即S4REN=0，则禁止接收。

S4TB8：在方式1，S4TB8为要发送的第9位数据，按需要由软件置位或清0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式0中，该位不用.

S4RB8：在方式1，S4RB8是接收到的第9位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式0中不用S4RB8(置S4SM2=0, S4RB8是接收到的停止位)。

S4TI：发送中断请求标志位。在停止位开始发送时由S4TI内部硬件置位，即S4TI=1,响应中断后S4TI必须用软件清零。

S4RI：接收中断请求标志位。在串行接收到停止位的中间时刻S4RI由内部硬件置位，即S4RI=1,向CPU发中断申请，响应中断后S4RI必须由软件清零。

S4CON的所有位可通过整机复位信号复位为全“0”。S4CON的字节地址为84H，不可位寻址。串行通信的中断请求：当一帧发送完成，内部硬件自动置位S4TI，即S4TI=1，请求中断处理；当接收完一帧信息时，内部硬件自动置位S4RI，即S4RI=1，请求中断处理。由于S4TI和S4RI以**“或逻辑”关系向主机请求中断，所以主机响应中断时事先并不知道是**S4TI还是S4RI请求的中断，必须在中断服务程序中查询S4TI和S4RI进行判别，然后分别处理。因此，两个中断请求标志位均不能由硬件自动置位，必须通过软件清0，否则将出现一次请求多次响应的错误。

2. 串行口4的数据缓冲寄存器S4BUF

STC15W4K60S4系列单片机的串行口4数据缓冲寄存器(S4BUF)的地址是85H，实际是2个缓冲器，写S4BUF的操作完成待发送数据的加载，读S4BUF的操作可获得已接收到的数据。两个操作分别对应两个不同的寄存器，1个是只写寄存器，1个是只读寄存器。

串行通道内设有数据寄存器。在所有的串行通信方式中，在写入S4BUF信号（MOV S4BUF,A）的控制下，把数据装入相同的9位移位寄存器，前面8位为数据字节，其最低位为移位寄存器的输出位。根据不同的工作方式会自动将“1”或S4TB8的值装入移位寄存器的第9位,并进行发送.

串行通道的接收寄存器是一个输入移位寄存器。在方式0和方式1时均为9位。当一帧接收完毕，移位寄存器中的数据字节装入串行数据缓冲器S4BUF中,其第9位则装入S4CON寄存器中的S4RB8位。如果由于S4SM2使得已接收到的数据无效时,S4RB8和S4BUF中内容不变.

由于接收通道内设有输入移位寄存器和S4BUF缓冲器，从而能使一帧接收完将数据由移位寄存器装入S4BUF后，可立即开始接收下一帧信息，主机应在该帧接收结束前从S4BUF缓冲器中将数据取走，否则前一帧数据将丢失。S4BUF以并行方式送往内部数据总线。

3. 串行口4既能选择定时器2作为其波特率发生器，也能选择定时器4作为其波特率发生器——定时器2的寄存器T2H,T2L和定时器4的寄存器T3H,T3L

定时器2寄存器T2H(地址为D6H，复位值为00H)及寄存器T2L(地址为D7H，复位值为00H)用于保存重装时间常数。

定时器4寄存器T4H(地址为D2H，复位值为00H)及寄存器T3L(地址为D3H，复位值为00H)用于保存重装时间常数。

STC MCU Limited

4. 定时器2的控制位——TR2、T2_C/$\overline{T}$、T2x12

AUXR：辅助寄存器（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2

T2R：定时器2允许控制位

0，不允许定时器2运行；

1，允许定时器2运行

T2_C/$\overline{T}$：控制定时器2用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T2/P3.1的外部脉冲进行计数)

T2x12：定时器2速度控制位

0，定时器2是传统8051速度，12分频；

1，定时器2的速度是传统8051的12倍，不分频

如果串口1或串口2用T2作为波特率发生器，则由T2x12决定串口1或串口2是12T还是1T.

5. 定时器4的控制位——TR4、T4_C/$\overline{T}$、T4x12

T4T3M(不可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/$\overline{T}$	T4x12	T4CLKO	T3R	T3_C/$\overline{T}$	T3x12	T3CLKO

B7 - T4R：定时器4运行控制位。

0，不允许定时器4运行；

1，允许定时器4运行。

B6 - T4_C/$\overline{T}$：控制定时器4用作定时器或计数器。

0，用作定时器(对内部系统时钟进行计数)；

1，用作计数器(对引脚T4/P0.7的外部脉冲进行计数)

B5 - T4x12：定时器4速度控制位。

0，定时器4速度是8051单片机定时器的速度，即12分频；

1，定时器4速度是8051单片机定时器速度的12倍，即不分频。

STC MCU Limited

6. 与串行口4中断相关的寄存器IE2

串行口4中断允许位ES4位于中断允许寄存器IE2中，中断允许寄存器的格式如下：

IE2：中断允许寄存器2（不可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4：定时器4的中断允许位。
1，允许定时器4产生中断；
0，禁止定时器4产生中断。

ET3：定时器3的中断允许位。
1，允许定时器3产生中断；
0，禁止定时器3产生中断。

ES4：串行口4中断允许位。
1，允许串行口4中断；
0，禁止串行口4中断

ES3：串行口3中断允许位。
1，允许串行口3中断；
0，禁止串行口3中断。

ET2：定时器2的中断允许位。
1，允许定时器2产生中断；
0，禁止定时器2产生中断。

ESPI：SPI中断允许位。
1，允许SPI中断；
0，禁止SPI中断。

ES2：串行口2中断允许位。
1，允许串行口2中断；
0，禁止串行口2中断。

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的总中断允许控制位
EA=1，CPU开放中断，
EA=0，CPU屏蔽所有的中断申请。

EA的作用是使中断允许形成多级控制。即各中断源首先受EA控制;其次还受各中断源自己的中断允许控制位控制。

STC MCU Limited

7. 串行口4在2组管脚之间切换的控制位——S4_S/P_SW2.2

通过设置寄存器P_SW2中的S4_S位，可以将串口4在2组管脚之间任意切换，P_SW2寄存器的格式如下：

P_SW2：外围设备切换控制寄存器2（不可位寻址）

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
P_SW2	BAH	外围设备功能切换控制寄存器2						S4_S	S3_S	S2_S	xxxx,x000

串口2/S2可在2个地方切换，由 S2_S 控制位来选择	
S2_S	S2可在P1/P4之间来回切换
0	串口2/S2在[P1.0/RxD2,P1.1/TxD2]
1	串口2/S2在[P4.6/RxD2_2,P4.7/TxD2_2]

串口3/S3可在2个地方切换，由 S3_S 控制位来选择	
S3_S	S3可在P0/P5之间来回切换
0	串口3/S3在[P0.0/RxD3,P0.1/TxD3]
1	串口3/S3在[P5.0/RxD3_2,P5.1/TxD3_2]

串口4/S4可在2个地方切换，由 S4_S 控制位来选择	
S4_S	S4可在P0/P5之间来回切换
0	串口4/S4在[P0.2/RxD4,P0.3/TxD4]
1	串口4/S4在[P5.2/RxD4_2,P5.3/TxD4_2]

STC MCU Limited

8.11 串行口4工作模式

——串口1和串口2的波特率相同时,串口1和串口2可共享T2作波特率发生器

STC15W4K60S4系列单片机的串行口4有两种工作模式，可通过软件编程对S4CON中的S4SM0的设置进行选择。其中模式0和模式1都为异步通信，每个发送和接收的字符都带有1个启动位和1个停止位。

8.11.1 串行口4的工作模式0----8位UART，波特率可变

10位数据通过RxD4/P0.2(RxD4_2/P5.2)接收，通过TxD4/P0.3(TxD4_2/P5.3)发送。一帧数据包含一个起始位(0)，8个数据位和一个停止位(1)。接收时，停止位进入特殊功能寄存器S4CON的S4RB8位。串行口4既可以选择定时器2作其波特率发生器，也可以选择定时器4作其波特率发生器。所以串行口4的波特率由定时器T2的溢出率或定时器T4的溢出率决定。

当串行口4选择定时器T2作为其波特率发生器(即S4ST4/S4SCON.1=0)时：

串口4在模式0的波特率 = 定时器T2的溢出率/4

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率=SYSclk / (65536 - [RL_TH2, RL_TL2])；

即此时，串行口4的波特率=SYSclk / (65536 - [RL_TH2, RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2, RL_TL2])；

即此时，串行口4的波特率=SYSclk / 12 / (65536 - [RL_TH2, RL_TL2]) / 4

上式中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。

当串行口4选择定时器T4作为其波特率发生器(即S4ST4/S4SCON.1=1)时：

串口4在模式0的波特率 = 定时器T4的溢出率/4

当T4工作在1T模式(T4T3M.5/T4x12=1)时，定时器4的溢出率=SYSclk / (65536 - [RL_TH4, RL_TL4])；

即此时，串行口4的波特率=SYSclk / (65536 - [RL_TH4, RL_TL4]) / 4

当T4工作在12T模式(T4T3M.5/T4x12=0)时，定时器4的溢出率=SYSclk /12/(65536 - [RL_TH4, RL_TL4])；

即此时，串行口4的波特率=SYSclk / 12 / (65536 - [RL_TH4, RL_TL4]) / 4

上式中RL_TH4是T4H的重装载寄存器，RL_TL4是T4L的重装载寄存器。

说明：当串口4和串口2的波特率相同时，串口4和串口2可以共享波特率发生器，此时建议用户选择定时器T2作为串口4的波特率发生器；当串口4和串口2的波特率不同时，才建议选择定时器T4作为串口4的波特率发生器(因串口2固定使用定时器T2作波特率发生器)。

STC-MCU Limited

8.11.2 串行口4的工作模式1----9位UART，波特率可变

11位数据通过TxD4/P0.3(TxD4_2/P5.3)发送，通过RxD4/P0.2(RxD4_2/P5.2)接收。一帧数据包含一个起始位(0)，8个数据位，一个可编程的第9位，和一个停止位(1)。发送时，第9位数据位来自特殊功能寄存器S4CON的S4TB8位. 接收时，第9位进入特殊功能寄存器S4CON的S4RB8位。串行口4既可以选择定时器2作其波特率发生器，也可以选择定时器4作其波特率发生器。所以串行口4的波特率由定时器T2的溢出率或定时器T4的溢出率决定。

当串行口4选择定时器T2作为其波特率发生器(即S4ST4/S4SCON. 1=0)时：

串口4在模式1的波特率 = 定时器T2的溢出率/4

当T2工作在1T模式(AUXR.2/T2x12=1)时，定时器2的溢出率=SYSclk / (65536 - [RL_TH2, RL_TL2])；

即此时，串行口4的波特率=SYSclk / (65536 - [RL_TH2, RL_TL2]) / 4

当T2工作在12T模式(AUXR.2/T2x12=0)时，定时器2的溢出率=SYSclk /12/(65536 - [RL_TH2, RL_TL2])；

即此时，串行口4的波特率=SYSclk / 12 / (65536 - [RL_TH2, RL_TL2]) / 4

上式中RL_TH2是T2H的重装载寄存器，RL_TL2是T2L的重装载寄存器。

当串行口4选择定时器T4作为其波特率发生器(即S4ST4/S4SCON. 1=1)时：

串口4在模式1的波特率= 定时器T4的溢出率/4

当T4工作在1T模式(T4T3M.5/T4x12=1)时，定时器4的溢出率=SYSclk / (65536 - [RL_TH4, RL_TL4])；

即此时，串行口4的波特率=SYSclk / (65536 - [RL_TH4, RL_TL4]) / 4

当T4工作在12T模式(T4T3M.5/T4x12=0)时，定时器4的溢出率=SYSclk /12/(65536 - [RL_TH4, RL_TL4])；

即此时，串行口4的波特率=SYSclk / 12 / (65536 - [RL_TH4, RL_TL4]) / 4

上式中RL_TH4是T4H的重装载寄存器，RL_TL4是T4L的重装载寄存器。

可见，模式1和模式0一样，其波特率可通过软件对定时器2的设置进行波特率的选择，是可变的。

说明：当串口4和串口2的波特率相同时，串口4和串口2可以共享波特率发生器，此时建议用户选择定时器T2作为串口4的波特率发生器；当串口4和串口2的波特率不同时，才建议选择定时器T4作为串口4的波特率发生器(因串口2固定使用定时器T2作波特率发生器)。

用户在程序中如何具体使用串口4

1. 设置串口4的工作模式，S4CON寄存器中的S4SM0决定了串口4的2种工作模式
2. 设置串口4的波特率相应的寄存器：

   定时器4寄存器T4H / T4L
3. 启动定时器4，让T4R位为1，定时器4就立即开始计数。
4. 设置T4T3M.5/T4x12, 确定定时器4的速度
5. 打开串口4中断，设置相应的控制位是：

   ES4, EA
6. 如要串口4接收，将S4REN置1 即可

   如要串口4发送，将数据送入S4BUF即可，

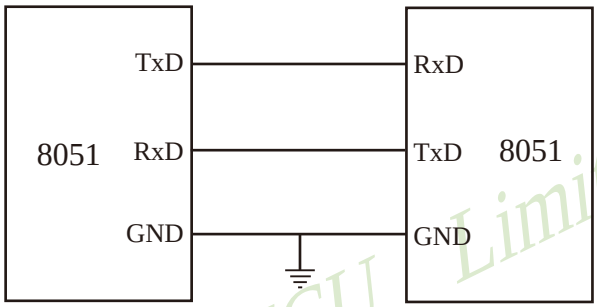
   接收完成标志S4RI, 发送完成标志S4TI, 要由软件清0。

STC MCU Limited

8.12 双机通信

STC15系列单片机的串行通信根据其应用可分为双机通信和多机通信两种。下面先介绍双机通信。

如果两个8051应用系统相距很近，可将它们的串行端口直接相连（TXD—RXD，RXD—TXD，GND—GND—地），即可实现双机通信。为了增加通信距离，减少通道及电源干扰，可采用RS—232C或RS—422、RS—485标准进行双机通信，两通信系统之间采用光—电隔离技术，以减少通道及电源的干扰，提高通信可靠性。



为确保通信成功，通信双方必须在软件上有系列的约定通常称为软件通信“协议”。现举例简介双机异步通信软件“协议”如下：

通信双方均选用2400波特的传输速率，设系统的主频SYSclk=6MHz,甲机发送数据，乙机接收数据。在双机开始通信时，先由甲机发送一个呼叫信号（例如“06H”），以询问乙机是否可以接收数据；乙机接收到呼叫信号后，若同意接收数据，则发回“00H”作为应答信号，否则发“05H”表示暂不能接收数据，；甲机只有在接收到乙机的应答信号“00H”后才可将存储在外部数据存储器中的内容逐一发送给乙机，否则继续向乙机发呼叫信号，直到乙机同意接收。其发送数据格式如下：

字节数n	数据1	数据2	数据3	…	数据n	累加校验和

字节数n：甲机向乙机发送的数据个数；

数据1~数据n：甲机将向乙机发送的n帧数据；

累加校验和：为字节数n、数据1、…、数据n,这(n+1)个字节内容的算术累加和.

乙机根据接收到的“校验和”判断已接收到的n个数据是否正确。若接收正确,向甲机回发“0FH”信号,否则回发“F0H”信号。甲机只有在接收到乙机发回的“0FH”信号才算完成发送任务，返回被调用的程序，否则继续呼叫，重发数据。

不同的通信要求，软件“协议”内容也不一样，有关需甲、乙双方共同遵守的约定应尽量完善，以防止通信不能正确判别而失败。

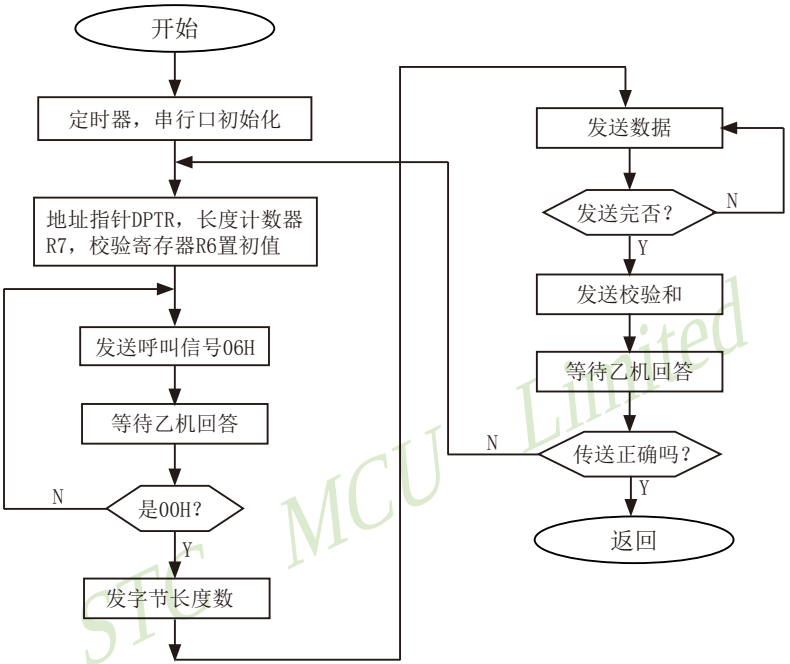
STC15系列单片机的串行通信，可直接采用查询法，也可采用自动中断法。

STC MCU Limited

（1）查询方式双机通信软件举例

①甲机发送子程序段

下图为甲机发送子程序流程图。



甲机发送程序设置：

(a) 波特率设置：选用定时器/计数器1定时模式、工作方式2，计数常数F3H，SMOD=1。波特率为2400（位/秒）；

(b) 串行通信设置：异步通信方式1，允许接收；

(c) 内部RAM和工作寄存器设置：

31H和30H单元存放发送的数据块首地址；

2FH单元存放发送的数据块个数；

R6为累加和寄存器。

甲机发送子程序清单：

```
START:
        MOV     TMOD，#20H              ；设置定时器/计数器1定时、工作方式2
        MOV     TH1，   #0F3H           ；设置定时计数常数
        MOV     TL1，   #0F3H           ；
        MOV     SCON，  #50H            ；串口初始化
        MOV     PCON，  #80H            ；设置SMOD=1
        SETB    TR1                     ；启动定时
ST-RAM:
        MOV     DPH，   31H             ；设置外部RAM数据指针
        MOV     DPL，   30H             ；DPTR初值
        MOV     R7，    2FH             ；发送数据块数送R7
        MOV     R6，    #00H            ；累加和寄存器R6清0
TX-ACK:
        MOV     A，     #06H            ；
        MOV     SBUF，  A               ；  } 发送呼叫信号“06H”
WAIT1:
        JBC     T1,     RX – YES        ；等待发送完呼叫信号
        SJMP    WAIT1                   ；未发送完转WATI1
RX-YES:
        JBC     RI，    NEXT1           ；判断乙机回答信号
        SJMP    RX-YES                  ；未收到回答信号，则等待
NEXT1:
        MOV     A，     SBUF            ；接收回答信号送A
        CJNE    A，     #00H,   TX-ACK  ；判断是否“00H”，否则重发呼叫信号
TX-BYT:
        MOV     A,      R7              ；  } 发送数据块数n
        MOV     SBUF,   A               ；
        ADD     A,      R6
        MOV     R6,     A
WAIT2:
        JBC     TI,     TX-NES          ；  } 等待发送完
        JMP     WAIT2                   ；
TX-NES:
        MOVX    A,      @DPTR           ；从外部RAM取发送数据
        MOV     SBUF,   A               ；发送数据块
        ADD     A,      R6
        MOV     R6,     A
        INC     DPTR                    ；DPTR指针加1
```

STC MCU Limited

```
WAIT3：
        JBC     TI,     NEXT2           ；判断一数据块发送完否
        SJMP    WAIT3                   ；等待发送完
NEXT2：
        DJNZ    R7,     TX-NES          ；判断发送全部结束否
TX-SUM：
        MOV     A,      R6              ；发送累加和给乙机
        MOV     SBUF，  A
WAIT4：
        JBC     TI,     RX-0FH          ；┐
        SJMP    WAIT4                   ；┘等待发送完
RX-0FH：
        JBC     RI,     IF-0FH          ；┐
        SJMP    RX-0FH                  ；┘等待接收乙机回答信号
IF-0FH：
        MOV     A,      SBUF；          ；┐
        CJNE    A,      #0FH,   ST-RAM  ；┘判断传输是否正确，否则重新发送
        RET                             ；返回
```

STC MCU Limited

乙机接收子程序段

接收程序段的设置：

（a） 波特率设置初始化：同发送程序；

（b） 串行通信初始化：同发送程序；

（c） 寄存器设置：

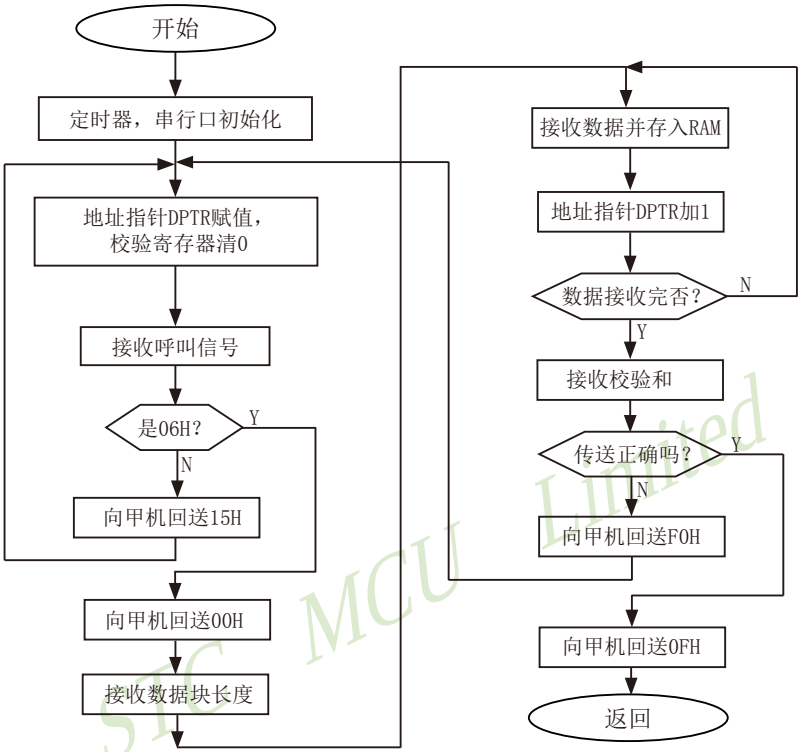
内部RAM 31H、30H单元存放接收数据缓冲区首地址。

R7——数据块个数寄存器。

R6——累加和寄存器。

（d） 向甲机回答信号：“0FH”为接收正确，“F0H”为传送出错，“00H”为同意接收数据，“05H”为暂不接收。

下图为双机通信查询方式乙机接收子程序流程图。



STC MCU Limited

接收子程序清单：

```
TART：
        MOV     TMOD，#20H
        MOV     TH1，   #0F3H           ；
                                        ｝定时器/计数器1设置
        MOV     TL1，   #0F3H           ；
        SETB    TR1                     ；启动定时器/计数器1
        MOV     SCON，  #50H            ；置串行通信方式1，允许接收
                                        ｝SMOD置位
        MOV     PCON，  #80H            ；
ST-RAM：
        MOV     DPH，   31H             ；
                                        ｝设置DPTR首地址
        MOV     DPL，   30H             ；
        MOV     R6，    #00H            ；校验和寄存器清0
```

```
RX-ACK：
        JBC     RI，    IF-06H          ；判断接收呼叫信号
        SJMP    RX-ACK                  ；等待接收呼叫信号
IF-06H：
        MOV     A，     SBUF            ；呼叫信号送A
        CJNEA   #06H，  TX-05H          ；判断呼叫信号正确否？
TX-00H：
        MOV     A，     #00H            ；
        MOV     SBUF，  A               ；  }向甲机发送“00H”，同意接收
WAIT1：
        JBC     TI，    RX-BYS          ；等待应答信号发送完
        SJMP    WAIT1
TX-05H：
        MOV     A，     #05H            ；向甲机发送“05H”呼叫
        MOV     SBUF，  A               ；不正确信号
WAIT2：
        JBC     TI，    HAVE1           ；等待发送完
        SJMP    WAIT2
HAVE1：
        LJMP    RX-ACK                  ；因呼叫错，返回重新接收呼叫
RX-BYS：
        JBC     RI，    HAVE2           ；等待接收数据块个数
        SJMP    RX-BYS                  ；
HAVE2：
        MOV     A，     SBUF            ；
        MOV     R7，    A               ；数据块个数帧送R7,R6
        MOV     R6，    A               ；
RX-NES：
        JBC     RI，    HAVE3           ；
        SJMP    RX-NES                  ；  }接收数据帧
HAVE3：
        MOV     A，     SBUF            ；
        MOVX    @DPTR，A                ；接收到的数据存入外部RAM
        INC     DPTR                    ；
        ADD     A，     R6              ；
        MOV     R6，    A               ；  }形成累加和
        DJNZ    R7，    RX-NES          ；判断数据是否接收完
```

STC MCU Limited

```
RX-SUM：
        JBC     RI，    HAVE4           ；
        SJMP    RX-SUM                  ；   等待接收校验和
HAVE4：
        MOV     A，     SBUF            ；
        CJNE    A，     R6，    TX-ERR  ；   判断传输是否正确
TX-RIT：
        MOV     A，     #0FH            ；
        MOV     SBUF，  A               ；   向甲机发送接收正确信息
WAIT3：
        JBC     TI，    GOOD            ；
        SJMP    WAIT3                   ；   等待发送结束
TX-ERR：
        MOV     A，     #0F0H           ；向甲机发送传输有误信号
        MOV     SBUF，  A
WAIT4：
        JBC     TI，    AGAIN           ；等待发送完
        SJMP    WAIT4
AGAIN：
        LJMP    ST-RAM                  ；返回重新开始接收
GOOD：
        RET                             ；传输正确返回
```

STC MCU Limited

（2）中断方式双机通信软件举例

在很多应用场合，双机通信的双方或一方采用中断方式以提高通信效率。由于STC15系列单片机的串行通信是双工的，且中断系统只提供一个中断矢量入口地址，所以实际上是中断和查询必须相结合，即接收/发送均可各自请求中断，响应中断时主机并不知道是谁请求中断，统一转入同一个中断矢量入口，必须由中断服务程序查询确定并转入对应的服务程序进行处理。

这里，任以上述协议为例，甲方（发送方）任以查询方式通信（从略），乙方（接收方）则改用中断—查询方式进行通信。

在中断接收服务程序中，需设置三个标志位来判断所接收的信息是呼叫信号还是数据块个数，是数据还是校验和。增设寄存器：内部RAM32H单元为数据块个数寄存器，33H单元为校验和寄存器，位地址7FH、7EH、7DH为标志位。

乙机接收中断服务程序清单

采用中断方式时，应在主程序中安排定时器/计数器、串行通信等初始化程序。通信接收的数据存放在外部RAM的首地址也需在主程序中确定。

主程序：

```
        ORG     0000H
        AJMP    START                   ；转至主程序起始处
        ORG     0023H
        LIMP    SERVE                   ；转中断服务程序处
        •
        •
        •
START：
        MOV     TMOD，#20H              ；定义定时器/计数器1定时、工作方式2
        MOV     TH1，   #0F3H           ；
        MOV     TL1，   #0F3H           ；  }设置波特率为2400位/秒
        MOV     SCON，  #50H            ；设置串行通信方式1，允许接收
        MOV     PCON，  #80H            ；设置SMOD=1
        SETB    TR1                     ；启动定时器
        SETB    7FH                     ；
        SETB    7EH                     ；设置标志位为1
        SETB    7DH                     ；
        MOV     31H，#10H               ；  规定接收的数据存储于外部RAM的
                                        ；  }起始地址1000H
        MOV     30H，#00H               ；
        MOV     33H，#00H               ；累加和单元清0
        SETB    EA                      ；  }开中断
        SETB    ES                      ；  }
        •
        •
        •
```

STC MCU Limited

中断服务程序：

```
SERVE：
        CLR     EA                  ；关中断
        CLR     RI                  ；清除接收中断请求标志
        PUSH    DPH                 ；
        PUSH    DPL                 ；现场保护
        PUSH    A                   ；
        JB      7FH，  RXACK        ；判断是否是呼叫信号
        JB      7EH，  RXBYS        ；判断是否是数据块数据
        JB      7DH，  RXDATA       ；判断是否是接收数据帧
RXSUM：
        MOV     A，    SBUF         ；接收到的校验和
        CJNE    A，  33H，  TXERR   ：判断传输是否正确
TXRI：
        MOV     A，    #0FH         ；
        MOV     SBUF，  A           ；  } 向甲机发送接收正确信号“0FH”
WAIT1：
        JNB     TI，   WAIT1        ；等待发送完毕
        CLR     TI                  ；清除发送中断请求标志位
        SJMP    AGAIN               ；转结束处理
TXERR：
        MOV     A，    #0F0H        ；
        MOV     SBUF，  A           ；  } 向甲机发送接收出错信号“F0H”
WAIT2：
        JNB     TI，   WAIT2        ；等待发送完毕
        CLR     TI                  ；清除发送中断请求标志
        SJMP    AGAIN               ；转结束处理
RXACK：
        MOV     A，    SBUF         ；判断是否是呼叫信号“06H”
        XRL     A，    #06H         ；异或逻辑处理
        JZ      TXREE               ；是呼叫，则转TXREE
TXNACK：
        MOV     A，    #05H         ；接收到的不是呼叫信号，则向甲机发送
        MOV     SBUF，  A           ；“05H”，要求重发呼叫
```

STC MCU Limited

```
WAIT3：
        JNB     TI，    WAIT3           ；  等待发送结束
        CLR     TI
        SJMP    RETURN                  ；转恢复现场处理
TXREE：
        MOV     A，     #00H            ；接收到的是呼叫信号，发送“00H”
        MOV     SBUF，  A               ；接收到的是呼叫信号，发送“00H”
WAIT4：
        JNB     TI，    WAIT4           ；等待发送完毕
        CLR     TI                      ；清除TI标志
        CLR     7FH                     ；清除呼叫标志
        SJMP    RETURN                  ；转恢复现场处理
RXBYS：
        MOV     A，     SBUF            ；接收到数据块数
        MOV     32H，   A               ；存入32H单元
        ADD     A，     33H             ；
        MOV     33H，   A               ；  形成累加和
        CLR     7EH                     ；清除数据块数标志
        SJMP    RETURN                  ；转恢复现场处理
RXDATA：
        MOV     DPH，   31H             ；
        MOV     DPL，   30H             ；  设置存储数据地址指针
        MOV     A，     SBUF            ；读取数据帧
        MOVX    @DPTR, A                ；将数据存外部RAM
        INC     DPTR                    ；地址指针加1
        MOV     31H，   DPH             ；
        MOV     30H，   DPL             ；  保存地址指针值
        ADD     A，     33H             ；
        MOV     33H，   A               ；  形成累加和
        DJNZ    32H，   RETURN          ；判断数据接收完否
        CLR     7DH                     ；清数据接收完标志
        SJMP    RETURN                  ；转恢复现场处理
```

STC MCU Limited

```
AGAIN：
        SETB    7FH             ；
        SETB    7EH             ；恢复标志位
        SETB    7DH             ；
        MOV     33H，   #00H    ；累加和单元清0
        MOV     31H，   #10H    ；
        MOV     30H，   #00H    ；恢复接收数据缓冲区首地址
RETURN：
        POP     A               ；
        POP     DPL             ；恢复现场
        POP     DPH             ；
        SETB    EA              ；开中断
        RET1                    ；返回
```

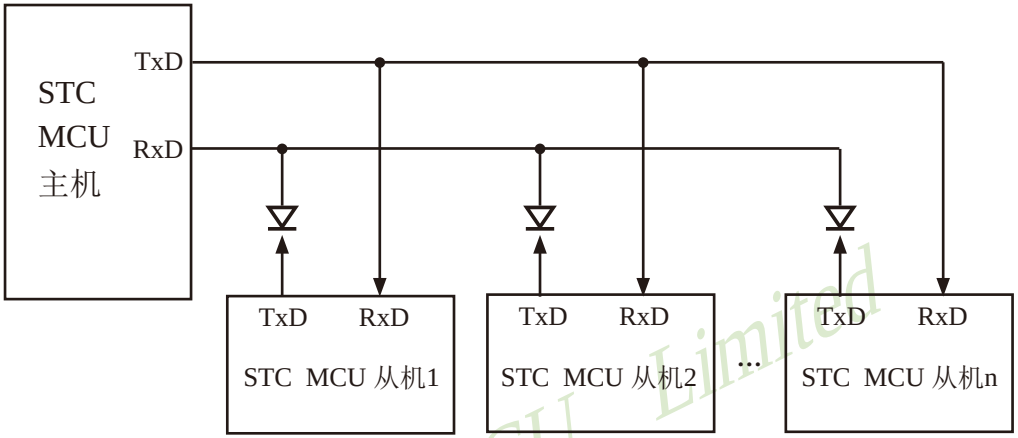
上述程序清单中，ORG为程序段说明伪指令，在程序汇编时，它向汇编程序说明该程序段的起始地址。

在实际应用中情况多种多样，而且是两台独立的计算机之间进行信息传输。因此，应周密考虑通信协议，以保证通信的正确性和成功率

STC MCU Limited

8.13 多机通信

在很多实际应用系统中，需要多台微计算机协调工作。STC15系列单片机的串行通信方式2和方式3具有多机通信功能，可构成各种分布式通信系统。下图为全双工主从式多机通信系统的连接框图。



上图为一台主机和几台从机组成的全双工多机通信系统。主机可与任一台从机通信，而从机之间的通信必须通过知己转发。

（1）多机通信的基本原理

在多机通信系统中，为保证主机（发送）与多台从机（接收）之间能可靠通信，串行通信必须具备识别能力。MCS-51系列单片机的串行通信控制寄存器SCON中设有多机通信选择位SM2。当程序设置SM2=1，串行通信工作于方式2或方式8，发送端通过对TB8的设置以区别于发送的是地址帧（TB8=1）还是数据帧（TB8=0），接收端通过对接收到RB8进行识别：当SM2=1，若接收到RB8=1，则被确认为呼叫地址帧，将该帧内容装入SBUF中，并置位RI=1，向CPU请求中断，进行地址呼叫处理；若RB8=0为数据帧，将不予理睬，接收的信息被丢弃。若SM2=0，则无论是地址帧还是数据帧均接收，并置位RI=1，向CPU请求中断，将该帧内容装入SBUF。据此原理，可实现多机通信。

对于上图的从机式多机通信系统，从机的地址为0，1，2，…，n。实现多机通信的过程如下：

① 置全部从机的SM2=1，处于只接收地址帧状态。

② 主机首先发送呼叫地址帧信息，将TB8设置为1，以表示发送的是 呼叫地址帧。

③ 所有从机接收到呼叫地址帧后，各自将接收到的主机呼叫的地址与本机的地址相比较：若比较结果相等，则为被寻址从机，清除SM2=0，准备接收从主机发送的数据帧，直至全部数据传输完；若比较不相等，则为非寻址从机，任维持SM2=1不变，对其后发来的数据帧不予理睬，即接收到的数据帧内容不装入SBUF，不置位，RI=0，不会产生中断请求，直至被寻址为止。

④ 主机在发送完呼叫地址帧后，接着发送一连串的数据帧，其中的

TB8=0，以表示为数据帧。

⑤ 当主机改变从机通信时间则再发呼叫地址帧，寻呼其他从机，原先被寻址的从机经分析得知主机在寻呼其他从机时，恢复其SM2=1，对其后主机发送的数据帧不予理睬。

上述过程均在软件控制下实现。

（2）多机通信协议简述

由于串行通信是在二台或多台各自完全独立的系统之间进行信息传输这就需要根据时间通信要求制定某些约定，作为通信规范遵照执行，协议要求严格、完善，不同的通信要求，协议的内容也不相同。在多机通信系统中要考虑的问题较多，协议内容比较复杂。这里仅例举几条作一说明。

上图的主从式多机通信系统，允许配置255台从机，各从机的地址分别为00H~FEH。

① 约定地址FFH为全部从机的控制命令，命令各从机恢复SM2=1状态，准备接收主机的地址呼叫。

② 主机和从机的联络过程约定：主机首先发送地址呼叫帧，被寻址的从机回送本机地址给主机，经验证地址相符后主机再向被寻址的从机发送命令字，被寻址的从机根据命令字要求回送本机的状态，若主机判断状态正常，主机即开始发送或接收数据帧，发送或接收的第一帧为传输数据块长度。

③ 约定主机发送的命令字为：

00H：要求从机接收数据块；

01H：要求从机发送数据块；

•
•
•

其他：非法命令。

④ 从机的状态字格式约定为：

B7	B6	B5	B4	B3	B2	B1	B0
ERR	0	0	0	0	0	TRDY	RRDY

定义：　若ERR=1，从机接收到非法命令；

若TRDY=1，从机发送准备就绪；

若RRDY=1，从机接收准备就绪；

⑤ 其他：如传输出错措施等。

STC MCU Limited

（3）程序举例

在实际应用中如传输波特率不太高，系统实时性有一定要求以及希望提高通信效率，则多半采用中断控制方式，但程序调试较困难，这就要求提高程序编制的正确性。采用查询方式，则程序调试较方便。这里仅以中断控制方式为例简单介绍主—从机之间一对一通信软件。

① 主机发送程序

该主机要发送的数据存放在内部RAM中，数据块的首地址为51H，数据块长度存放做50H单元中，有关发送前的初始化、参数设置等采用子程序格式，所有信息发送均由中断服务程序完成。当主机需要发送时，在完成发送子程序的调用之后，随即返回主程序继续执行。以后只需查询PSW·5的F0标志位的状态即可知道数据是否发送完毕。

要求主机向#5从机发送数据，中断服务程序选用工作寄存器区1的R0~R7。

主机发送程序清单：

```
        ORG     0000H
        AJMP    MAIN                    ；转主程序
        ORG     0023H                   ；发送中断服务程序入口
        LJMP    SERVE                   ；转中断服务程序
          :
          :
MAIN:   . . .   . . .                   ；主程序
          :
          :
        ORG     1000H                   ；发送子程序入口
TXCALL：
        MOV     TMOD，#20H              ；设置定时器/计数器1定时、方式2
        MOV     TH1，   #0F3H           ；设置波特率为2400位/秒
        MOV     TL1，   #0F3H           ；置位SMOD
        MOV     PCON，  #80H            ；
        SETB    TR1                     ；启动定时器/计数器1
        MOV     SCON，  #0D8H           ；串行方式8，允许接收，TB8=1
        SETB    EA                      ；开中断总控制位
        CLR     ES                      ；禁止串行通信中断
TXADDR：
        MOV     SBUF，  #05H            ；发送呼叫从机地址
WAIT1：
        JNB     TI，    WAIT1           ；等待发送完毕
        CLR     TI                      ；复位发送中断请求标志
```

STC MCU Limited

```
RXADDR：
        JNB     RI，    RXADDR          ；等待从机回答本机地址
        CLR     TI                      ；复位接收中断请求标志
        MOV     A，     SBUF            ；读取从机回答的本机地址
        CJNE    A，     #05H，  TXADDR  ；判断呼叫地址符否，否则重发
        CLR     TB8                     ；地址相符，复位TB8=0，准备发数据
        CLR     PSW. 5                  ；复位F0=0标志位
        MOV     08H，   #50H            ；发送数据地址指针送R0
        MOV     0CH，   50H             ；数据块长度送R4
        INC     0CH                     ；数据块长度加1
        SETB    ES                      ；允许串行通信中断
        RET                             ；返回主程序
          ：
SERVE：
        CLR     TI                      ；中断服务程序段，清中断请求标志TI
        PUSH    PSW                     ；
        PUSH    A                       ；  } 现场入栈保护
        CLR     RS1                     ；
        SETB    RS0                     ；  } 选择工作寄存器组1
TXDATA：
        MOV     SBUF，  @R0             ；发送数据块长度及数据
WAIT2：
        JNB     TI，    WAIT2           ；等待发送完毕
        CLR     TI                      ；复位TI=0
        INC     R0                      ；地址指针加1
        DJNZ    R4，    RETURN          ；数据块未发送完，转返回
        SETB    PSW. 5                  ；已发送完毕置位F0=1
        CLR     ES                      ；关闭串行中断
RETURN：
        POP     A                       ；
        POP     PSW                     ；  } 恢复现场
        RETI                            ；返回
```

STC MCU Limited

②从机接收程序

主机发送的地址呼叫帧，所有的从机均接收，若不是呼叫本机地址即从中断返回；若是本机地址，则回送本机地址给主机作为应答，并开始接收主机发送来的数据块长度帧，并存放于内部RAM的60H单元中，紧接着接收的数据帧存放于61H为首地址的内部RAM单元中，程序中还选用20·0H、20·1H位作标志位，用来判断接收的是地址、数据块长度还是数据，选用了2FH、2EH两个字节单元用于存放数据字节数和存储数据指针。#5从机的接收程序如下，供参考。

#5从机接收程序清单：

```
        ORG     0000H
        AJMP    START                   ；转主程序段
        ORG     0023H
        LJMP    SERVE                   ；从中断入口转中断服务程序
        ORG     0100H
START：
        MOV     TMOD，#20H              ；主程序段：初始化程序，设置定时
        MOV     TH1，   #0F3H           ；器/计数器1定时、工作方式2，设
        MOV     TL1，   #0F3H           ；置波特率为2400位/秒的有关初值
        MOV     PCON，  #80H            ；置位SMOD
        MOV     SCON，  #0F0H           ；设置串行方式3，允许接收，SM2=1
        SETB    TR1                     ；启动定时器/计数器1
        SETB    20·0                    ；
        SETB    20·1                    ；  置标志位为1
        SETB    EA                      ；
        SETB    ES                      ；  开中断
        ：
        ORG     1000H
SERVE：
        CLR     RI                      ；清接收中断请求标志RI=0
        PUSH    A                       ；
        PUSH    PSW                     ；  现场保护
        CLR     RS1                     ；
        SETB    RS0                     ；  选择工作寄存器组1
        JB      20·0H，  ISADDR         ；判断是否是地址帧
        JB      20·1H，  ISBYTE         ；判断是否是数据块长度帧
```

STC MCU Limited

```
ISDATA：
        MOV     R0，    2EH                     ；数据指针送R0
        MOV     A，     SBUF                    ；接收数据
        MOV     @R0，   A
        INC     2EH                             ；数据指针加1
        DJNZ    2FH，   RETURN                  ；判断数据接收完否？
        SETB    20・0H                          ；
        SETB    20・1H                          ；恢复标志位
        SETB    SM2                             ；
        SJMP    RETURN                          ；转入恢复现场，返回
ISADDR：
        MOV     A，     SBUF                    ；  是地址呼叫，判断与本机地址
        CJNE    A，     #05H，  RETURN          ；} 相符否，不符则转返回
        MOV     SBUF，  #01H                    ；相符，发回答信号“01H”
WAIT：
        JNB     TI，    WAIT                    ；等待发送结束
        CLR     TI                              ；清0TI，20・0，SM2
        CLR     20・0H                          ；清0TI，20・0，SM2
        CLR     SM2                             ；清0TI，20・0，SM2
        SJMP    RETURN                          ；转返回
ISBYTES：
        MOV     A，     SBUF                    ；接收数据块长度帧
        MOV     R0，    #60H                    ；
        MOV     @R0，   A                       ；将数据块长度存入内部RAM
        MOV     2FH，   A                       ；60H单元及2FH单元
        MOV     2EH，   #61H                    ；置首地址61H于2EH单元
        CLR     20・1H                          ；清20・1H标志，表示以后接收的为数据
RETURN：
        POP     PSW                             ；} 恢复现场
        POP     A                               ；
        RETI                                    ；返回
```

多机通信方式可多种多样，上例仅以最简单的住一从式作了简单介绍，仅供参考。

对于串行通信工作方式0的同步方式，常用于通过移位寄存器进行扩展并行I/O口，或配置某些串行通信接口的外部设备。例如，串行打印机、显示器等。这里就不一一举例了。

STC MCU Limited

8.14 串口1作为增强型串口使用时的自动地址识别功能

8.14.1 与串口1自动地址识别功能相关的特殊功能寄存器

符号	描述	地址	位地址及符号 MSB LSB								复位值
SCON	串口控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
SBUF	串口发送/接收数据缓冲器	99H									xxxx xxxxB
SADEN	从机地址屏蔽位寄存器	B9H									0000 0000B
SADDR	从机地址寄存器	A9H									0000 0000B

1. 串行口1的控制寄存器SCON

串行控制寄存器SCON用于选择串行通信的工作方式和某些控制功能。其格式如下：

SCON：串行控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE：当PCON寄存器中的SMOD0/PCON.6位为1时，该位用于帧错误检测。当检测到一个无效停止位时，通过UART接收器设置该位。它必须由软件清零。

当PCON寄存器中的SMOD0/PCON.6位为0时，该位和SM1一起指定串行通信的工作方式，如下表所示。

其中SM0、SM1按下列组合确定串行口1的工作方式：

SM0	SM1	工作方式	功能说明	波特率
0	0	方式0	同步移位串行方式：移位寄存器	当UART_M0x6 = 0时，波特率是SYSclk/12, 当UART_M0x6 = 1时，波特率是SYSclk / 2
0	1	方式1	8位UART，波特率可变	串行口1用定时器1作为其波特率发生器且定时器1工作于模式0(16位自动重装载模式)或串行口用定时器2作为其波特率发生器时，波特率=(定时器1的溢出率或定时器T2的溢出率)/4. 注意：此时波特率与SMOD无关。 当串行口1用定时器1作为其波特率发生器且定时器1工作于模式2(8位自动重装模式)时，波特率=(2^{SMOD}/32)×(定时器1的溢出率)
1	0	方式2	9位UART	(2^{SMOD} / 64) x SYSclk系统工作时钟频率
1	1	方式3	9位UART，波特率可变	当串行口1用定时器1作为其波特率发生器且定时器1工作于模式0(16位自动重装载模式)或串行口用定时器2作为其波特率发生器时，波特率=(定时器1的溢出率或定时器T2的溢出率)/4. 注意：此时波特率与SMOD无关。 当串行口1用定时器1作为其波特率发生器且定时器1工作于模式2(8位自动重装模式)时，波特率=(2^{SMOD}/32)×(定时器1的溢出率)

SM2：允许方式2或方式3多机通信控制位。
在方式2或方式3时，如果SM2位为1且REN位为1，则接收机处于地址帧筛选状态。此时可以利用接收到的第9位(即RB8)来筛选地址帧：若RB8=1，说明该帧是地址帧，地址信息可以进入SBUF，并使RI为1，进而在中断服务程序中再进行地址号比较；若RB8=0，说明该帧不是地址帧，应丢掉且保持RI=0。在方式2或方式3中，如果SM2位为0且REN位为1，接收收机处于地址帧筛选被禁止状态。不论收到的RB8为0或1，均可使接收到的信息进入SBUF,并使RI=1,此时RB8通常为校验位.
方式1和方式0是非多机通信方式，在这两种方式时，要设置SM2 应为0。

REN：允许/禁止串行接收控制位。由软件置位REN，即REN=1为允许串行接收状态，可启动串行接收器RxD，开始接收信息。软件复位REN，即REN=0，则禁止接收。

TB8：在方式2或方式3，它为要发送的第9位数据，按需要由软件置位或清0。例如，可用作数据的校验位或多机通信中表示地址帧/数据帧的标志位。在方式0和方式1中，该位不用.

RB8：在方式2或方式3，是接收到的第9位数据，作为奇偶校验位或地址帧/数据帧的标志位。方式0中不用RB8(置SM2=0). 方式1中也不用RB8(置SM2=0, RB8是接收到的停止位)。

TI：发送中断请求标志位。在方式0，当串行发送数据第8位结束时，由内部硬件自动置位，即TI=1，向主机请求中断，响应中断后TI必须用软件清零，即TI=0。在其他方式中，则在停止位开始发送时由内部硬件置位，即TI=1,响应中断后TI必须用软件清零。

RI：接收中断请求标志位。在方式0，当串行接收到第8位结束时由内部硬件自动置位RI=1，向主机请求中断，响应中断后RI必须用软件清零，即RI=0。在其他方式中，串行接收到停止位的中间时刻由内部硬件置位，即RI=1,向CPU发中断申请，响应中断后RI必须由软件清零。

SCON的所有位可通过整机复位信号复位为全“0”。SCON的字节地址为98H，可位寻址，各位地址为98H~~9FH，可用软件实现位设置。

串行通信的中断请求：当一帧发送完成，内部硬件自动置位TI，即TI=1，请求中断处理；当接收完一帧信息时，内部硬件自动置位RI，即RI=1，请求中断处理。由于TI和RI以**“或逻辑”关系向主机请求中断，所以主机响应中断时事先并不知道是**TI还是RI请求的中断，必须在中断服务程序中查询TI和RI进行判别，然后分别处理。因此，两个中断请求标志位均不能由硬件自动置位，必须通过软件清0，否则将出现一次请求多次响应的错误。

STC MCU Limited

2. 串行口数据缓冲寄存器SBUF

STC15系列单片机的串行口1缓冲寄存器(SBUF)的地址是99H，实际是2个缓冲器，写SBUF的操作完成待发送数据的加载，读SBUF的操作可获得已接收到的数据。两个操作分别对应两个不同的寄存器，1个是只写寄存器，1个是只读寄存器。

串行通道内设有数据寄存器。在所有的串行通信方式中，在写入SBUF信号(MOV　SBUF,A)的控制下，把数据装入相同的9位移位寄存器，前面8位为数据字节，其最低位为移位寄存器的输出位。根据不同的工作方式会自动将“1”或TB8的值装入移位寄存器的第9位,并进行发送.

串行通道的接收寄存器是一个输入移位寄存器。在方式0时它的字长为8位，其他方式时为9位。当一帧接收完毕，移位寄存器中的数据字节装入串行数据缓冲器SBUF中,其第9位则装入SCON寄存器中的RB8位。如果由于SM2使得已接收到的数据无效时,RB8和SBUF中内容不变.

由于接收通道内设有输入移位寄存器和SBUF缓冲器，从而能使一帧接收完将数据由移位寄存器装入SBUF后，可立即开始接收下一帧信息，主机应在该帧接收结束前从SBUF缓冲器中将数据取走，否则前一帧数据将丢失。SBUF以并行方式送往内部数据总线。

STC MCU Limited

3. 从机地址控制寄存器SADEN和SADDR

为了方便多机通信，STC15系列单片机设置了从机地址控制寄存器SADEN和SADDR。其中SADEN是从机地址掩模寄存器(地址为B9H，复位值为00H)，SADDR是从机地址寄存器(地址为A9H，复位值为00H)。

8.14.2 串口1自动地址识别功能的介绍

自动地址识别功能典型应用在多机通讯领域，其主要原理是从机系统通过硬件比较功能来识别来自于主机串口数据流中的地址信息，通过寄存器SADDR和SADEN设置的本机的从机地址，硬件自动对从机地址进行过滤，当来自于主机的从机地址信息与本机所设置的从机地址相匹配时，硬件产生串口中断；否则硬件自动丢弃串口数据，而不产生中断。当众多处于空闲模式的从机链接在一起时，只有从机地址相匹配的从机才会从空闲模式唤醒，从而可以大大降低从机MCU的功耗，即使从机处于正常工作状态也可避免不停地进入串口中断而降低系统执行效率。

要使用串口的自动地址识别功能，首先需要将参与通讯的MCU的串口通讯模式设置为模式2或者模式3（通常都选择波特率可变的模式3，因为模式2的波特率是固定的，不便于调节），并开启从机的SCON的SM2位。对于串口模式2或者模式3的9位数据位中，第9位数据（存放在RB8中）为地址/数据的标志位，当第9位数据为1时，表示前面的8位数据（存放在SBUF中）为地址信息。当SM2＝1时，从机MCU会自动过滤掉非地址数据（第9位为0的数据），而对SBUF中的地址数据（第9位为1的数据）自动与SADDR和SADEN所设置的本机地址进行比较，若地址相匹配，则会将RI置“1”，并产生中断，否则不予处理本次接收的串口数据。

从机地址的设置是通过SADDR和SADEN两个寄存器进行设置的。SADDR为从机地址寄存器，里面存放本机的从机地址。SADEN为从机地址屏蔽位寄存器，用于设置地址信息中的“don’t care bit”（忽略位），设置方法如下：

例如

SADDR ＝ 11001010

SADEN ＝ 10000001

则匹配地址为　1xxxxxx0

即，只要主机送出的地址数据中的bit0为0且bit7为1就可以和本机地址相匹配

再例如

SADDR ＝ 11001010

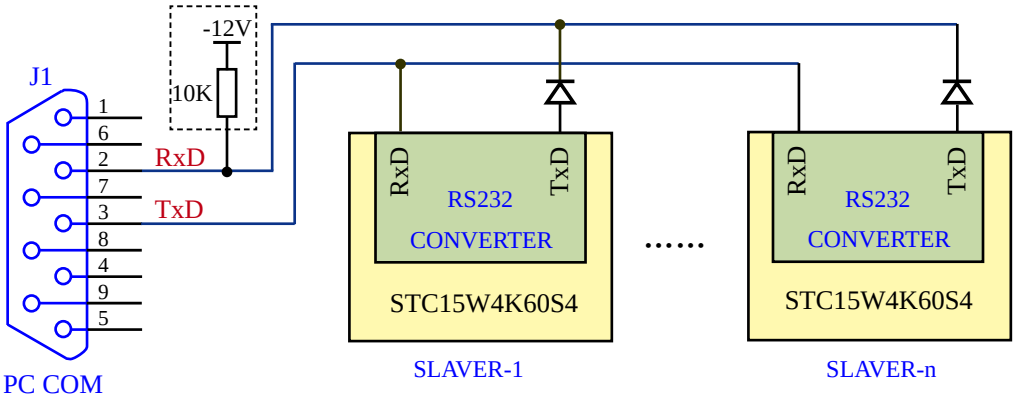
SADEN ＝ 00001111

则匹配地址为　xxxx1010

即，只要主机送出的地址数据中的低4位为1010就可以和本机地址相匹配，而高4为被忽略，可以为任意值。

主机可以使用广播地址（FFH）同时选中所有的从机来进行通讯。

STC MCU Limited

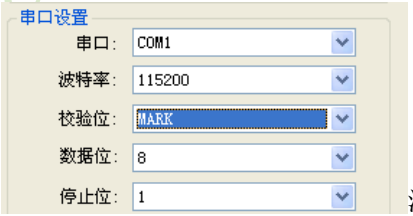


示例代码的测试方法:

1、 首先将两个MCU按照上图的链接方法与电脑相连接

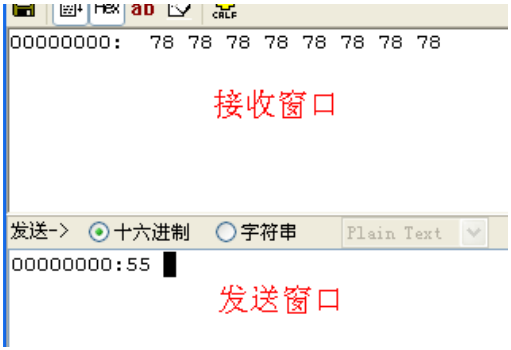
2、 将代码中SLAVE定义为0(“#define SLAVER 0”),编译产生的HEX文件烧录到SLAVER-1的MCU中;然后将SLAVE定义为1(“#define SLAVER 1”),编译产生的HEX文件烧录到SLAVER-2的MCU中

3、 在PC端,打开串口助手,将串口设置如有图,

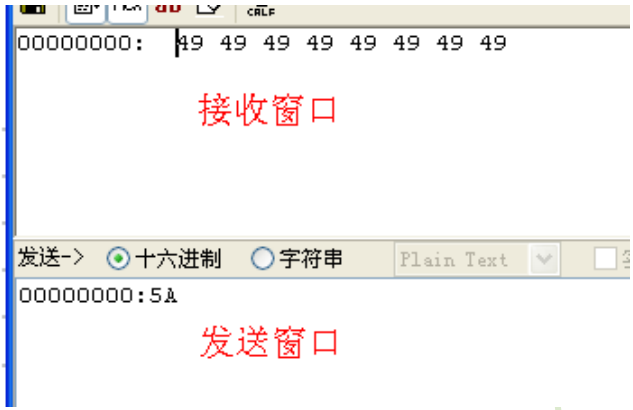


注意校验位

4、 在串口助手终端,发送0x55,则会选中从机1,此时从机1应答的数据为8个0x78



5、 串口助手终端再发送0x5a, 则会选中从机2, 此时从机2应答的数据为8个0x49，如图



8.14.3 串口1自动地址识别功能的测试程序(C和汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 串口1地址自动识别举例举例-----------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef   unsigned   char     BYTE;
typedef   unsigned   int      WORD;

//-----------------------------------------

#define   SLAVER          0             //定义从机编号,0为从机1, 1为从机2

#if       SLAVER   ==     0
#define   SAMASK          0x33          //从机1地址屏蔽位
```

```
#define   SERADR          0x55          //从机1的地址为xx01,xx01
#define   ACKTST          0x78          //从机1应答测试数据
#else
#define   SAMASK          0x3C          //从机2地址屏蔽位
#define   SERADR          0x5A          //从机2的地址为xx01,10xx
#define   ACKTST          0x49          //从机2应答测试数据
#endif

#define   URMD   0              //0:使用定时器2作为波特率发生器
                                //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器

sfr       T2H     =     0xd6;         //定时器2高8位
sfr       T2L     =     0xd7;         //定时器2低8位

sfr       AUXR    =     0x8e;         //辅助寄存器

sfr       SADDR   =     0xA9;         //从机地址寄存器
sfr       SADEN   =     0xB9;         //从机地址屏蔽寄存器

void InitUart();

char count;

void main()
{
        InitUart();                   //初始化串口
        ES = 1;
        EA = 1;
        while (1);
}

/*----------------------------
UART 中断服务程序
-----------------------------*/
void Uart() interrupt 4 using 1
{
        if (TI)
        {
                TI = 0;               //清除TI位
                if (count != 0)
                {
                        count--;
                        SBUF =  ACKTST;         //继续发送应答数据
                }
                else
                {
                        SM2  =   1;             //若发送完成,则重新开始地址检测
                }
        }
```

STC MCU Limited

```
        if (RI)
        {
                RI      =       0;              //清除RI位
                SM2     =       0;              //本机被选中后,进入数据接收状态
                count   =       7;
                SBUF    =       ACKTST;         //并开发送应答数据
        }
}

/*----------------------------
初始化串口
----------------------------*/
void InitUart()
{
        SADDR   =       SERADR;
        SADEN   =       SAMASK;
        SCON    =       0xf8;           //设置串口为9位可变波特率,使能多机通讯检测,
                                        //(将TB8设置为1,方便与PC直接通讯测试)
#if     URMD    ==      0
        T2L     =       0xd8;           //设置波特率重装值
        T2H     =       0xff;           //115200 bps(65536-18432000/4/115200)
        AUXR    =       0x14;           //T2为1T模式, 并启动定时器2
        AUXR    |=      0x01;           //选择定时器2为串口1的波特率发生器
#elif   URMD    ==      1
        AUXR    =       0x40;           //定时器1为1T模式
        TMOD    =       0x00;           //定时器1为模式0(16位自动重载)
        TL1     =       0xd8;           //设置波特率重装值
        TH1     =       0xff;           //115200 bps(65536-18432000/4/115200)
        TR1     =       1;              //定时器1开始启动
#else
        TMOD    =       0x20;           //设置定时器1为8位自动重装载模式
        AUXR    =       0x40;           //定时器1为1T模式
        TH1 =   TL1 =   0xfb;           //115200 bps(256 - 18432000/32/115200)
        TR1     =       1;
#endif
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 串口1地址自动识别举例举例-----------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可 */
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define   SLAVER          0             //定义从机编号,0为从机1, 1为从机2

#if       SLAVER ==       0
#define   SAMASK          0x33          //从机1地址屏蔽位
#define   SERADR          0x55          //从机1的地址为xx01,xx01
#define   ACKTST          0x78          //从机1应答测试数据
#else
#define   SAMASK          0x3C          //从机2地址屏蔽位
#define   SERADR          0x5A          //从机2的地址为xx01,10xx
#define   ACKTST          0x49          //从机2应答测试数据
#endif

#define   URMD    0                     //0:使用定时器2作为波特率发生器
                                        //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                        //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
T2H       DATA    0D6H                  //定时器2高8位
T2L       DATA    0D7H                  //定时器2低8位
AUXR      DATA    08EH                  //辅助寄存器

SADDR     DATA    0A9H                  //从机地址寄存器
SADEN     DATA    0B9H                  //从机地址屏蔽寄存器

COUNT     DATA    20H
//----------------------------------------
          ORG     0000H
          LJMP    MAIN

          ORG     0023H
          LJMP    UART_ISR
//----------------------------------------
          ORG     0100H
MAIN:
          MOV     SP,     #3FH
```

STC MCU Limited

```
        LCALL   INIT_UART               //初始化串口
        SETB    ES
        SETB    EA
        SJMP    $

//---------------------------------------
//串口中断服务程序
UART_ISR:
        PUSH    PSW
        PUSH    ACC
        JNB     TI,     CHK_RX
        CLR     TI
        MOV     A,      COUNT           //发送完成8个数据后,就不再发送
        JZ      RESTART
        DEC     COUNT
        MOV     SBUF,   #ACKTST         //发送应答测试数据
        JMP     UREXIT
RESTART:
        SETB    SM2                     //若发送完成,则重新开始地址检测
        JMP     UREXIT
CHK_RX:
        JNB     RI,     UREXIT
        CLR     RI
        CLR     SM2                     //本机被选中后,进入数据接收状态
        MOV     SBUF,   #ACKTST         //并开发送应答数据
        MOV     COUNT,  #7
UREXIT:
        POP     ACC
        POP     PSW
        RETI

/*---------------------------
初始化串口
---------------------------*/
INIT_UART:
        MOV     SADDR,  #SERADR
        MOV     SADEN,  #SAMASK
        MOV     SCON,   #0F8H           //设置串口为9位可变波特率,使能多机通讯检测,
                                        //(将TB8设置为1,方便与PC直接通讯测试)
#if     URMD    ==      0
        MOV     T2L,    #0D8H           //设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H            //T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H            //选择定时器2为串口1的波特率发生器
#elif   URMD    ==      1
        MOV     AUXR,   #40H            //;定时器1为1T模式
        MOV     TMOD,   #00H            //定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H           //设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
```

STC MCU Limited

```
        SETB    TR1                     //定时器1开始运行
#else
        MOV     TMOD,   #20H            //设置定时器1为8位自动重装载模式
        MOV     AUXR,   #40H            //定时器1为1T模式
        MOV     TL1,    #0FBH           //115200 bps(256 - 18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1
#endif
        RET

//-----------------------------------------

        END
```

STC MCU Limited

8.15 串行口1的中继广播方式

串行口1可在3组管脚间进行切换：[RxD/P3.0, TxD/P3.1]；[RxD_2/P3.6, TxD_2/P3.7]；[RxD_3/P1.6, TxD_3/P1.7].

Mnemonic	Add	Name	7	BB6	B5	B4	B3	B2	B1	B0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,0000

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

串口1的RxD管脚和TxD管脚可以在3组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1]；
[RxD_2/P3.6, TxD_2/P3.7]；
[RxD_3/P1.6, TxD_3/P1.7].

串行口1的中继广播方式除可以在用户程序中设置Tx_Rx/CLK_DIV.4来选择外，还可以在STC-ISP下载编程软件中设置。

当单片机的工作电压低于上电复位门槛电压(POR，3V单片机在1.8V附近，5V单片机在3.2V附近)时，Tx_Rx默认为0，即串行口1默认为正常工作方式。当单片机的工作电压高于上电复位门槛电压(POR，3V单片机在1.8V附近，5V单片机在3.2V附近)时，单片机首先读取用户在STC-ISP下载编程软件中的设置，如果用户允许了“单片机TxD管脚的对外输出实时反映RxD端口输入的电平状态”即中继广播方式，则上电复位后P3.7/TxD_2管脚的对外输出可以实时反映P3.6/RxD_2端口输入的电平状态，如果用户未选择“单片机TxD管脚的对外输出实时反映RxD端口输入的电平状态”，则上电复位后串口1为正常工作方式，即P3.7/TxD_2管脚的对外输出不实时反映P3.6/RxD_2端口输入的电平状态。

串行口1的位置和中继广播方式除可以在STC-ISP下载编程软件中设置外，还可以在用户的用户程序中用设置。在STC-ISP下载编程软件中的设置是在单片机上电复位后就可以执行的，如果用户在用户程序中的设置与STC-ISP下载编程软件中的设置不一致时，当执行到相应的用户程序时就会覆盖原来STC-ISP下载编程软件中的设置。

STC MCU Limited

8.16 用T0软件模拟串行口的测试程序(C及汇编)

——如串行口不够用或无串行口可用[P3.0, P3.1]结合定时器0软件模拟串行口

1. C程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15Fxx 系列 软件模拟串口举例------------------------------------------------------------*/
/* --- 研发顾问QQ：800003751---------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8058芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
//-----------------------------------------
//define baudrate const
//BAUD = 256 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
//NOTE: (FOSC/3/BAUDRATE) must be greater then 98, (RECOMMEND GREATER THEN 110)

//#define  BAUD    0xF400                 // 1200bps @ 11.0592MHz
//#define  BAUD    0xFA00                 // 2400bps @ 11.0592MHz
//#define  BAUD    0xFD00                 // 4800bps @ 11.0592MHz
//#define  BAUD    0xFE80                 // 9600bps @ 11.0592MHz
//#define  BAUD    0xFF40                 //19200bps @ 11.0592MHz
//#define  BAUD    0xFFA0                 //38400bps @ 11.0592MHz

//#define  BAUD    0xEC00                 // 1200bps @ 18.432MHz
//#define  BAUD    0xF600                 // 2400bps @ 18.432MHz
//#define  BAUD    0xFB00                 // 4800bps @ 18.432MHz
//#define  BAUD    0xFD80                 // 9600bps @ 18.432MHz
//#define  BAUD    0xFEC0                 //19200bps @ 18.432MHz
#define   BAUD    0xFF60                 //38400bps @ 18.432MHz

//#define  BAUD    0xE800                 // 1200bps @ 22.1184MHz
//#define  BAUD    0xF400                 // 2400bps @ 22.1184MHz
//#define  BAUD    0xFA00                 // 4800bps @ 22.1184MHz
//#define  BAUD    0xFD00                 // 9600bps @ 22.1184MHz
```

STC MCU Limited

```
//#define  BAUD    0xFE80                    //19200bps @ 22.1184MHz
//#define  BAUD    0xFF40                    //38400bps @ 22.1184MHz
//#define  BAUD    0xFF80                    //57600bps @ 22.1184MHz
sfr         AUXR    =       0x8E;
sbit        RXB     =       P3^0;           //define UART TX/RX port
sbit        TXB     =       P3^1;

typedef bit     BOOL;
typedef unsigned char    BYTE;
typedef unsigned int     WORD;

BYTE    TBUF,   RBUF;
BYTE    TDAT,   RDAT;
BYTE    TCNT,   RCNT;
BYTE    TBIT,   RBIT;
BOOL    TING,   RING;
BOOL    TEND,   REND;

void   UART_INIT();

BYTE    t, r;
BYTE    buf[16];

void main()
{
        TMOD = 0x00;                        //timer0 in 16-bit auto reload mode
        AUXR = 0x80;                        //timer0 working at 1T mode
        TL0 = BAUD;
        TH0 = BAUD>>8;                      //initial timer0 and set reload value
        TR0 = 1;                            //tiemr0 start running
        ET0 = 1;                            //enable timer0 interrupt
        PT0 = 1;                            //improve timer0 interrupt priority
        EA = 1;                             //open global interrupt switch

        UART_INIT();

        while (1)
        {                                   //user's function
                if (REND)
                {
                        REND = 0;
                        buf[r++ & 0x0f] = RBUF;
                }
                if (TEND)
```

STC MCU Limited

```
                {
                        if (t != r)
                        {
                                TEND = 0;
                                TBUF = buf[t++ & 0x0f];
                                TING = 1;
                        }
                }
        }
}

//-----------------------------------------
//Timer interrupt routine for UART

void tm0() interrupt 1 using 1
{
        if (RING)
        {
                if (--RCNT == 0)
                {
                        RCNT = 3;                       //reset send baudrate counter
                        if (--RBIT == 0)
                        {
                                RBUF = RDAT;            //save the data to RBUF
                                RING = 0;               //stop receive
                                REND = 1;               //set receive completed flag
                        }
                        else
                        {
                                RDAT >>= 1;
                                if (RXB) RDAT |= 0x80;  //shift RX data to RX buffer
                        }
                }
        }
        else if (!RXB)
        {
                RING = 1;                       //set start receive flag
                RCNT = 4;                       //initial receive baudrate counter
                RBIT = 9;                       //initial receive bit number (8 data bits + 1 stop bit)
        }

        if (--TCNT == 0)
        {
                TCNT = 3;                               //reset send baudrate counter
```

STC MCU Limited

```
                if (TING)                                   //judge whether sending
                {
                        if (TBIT == 0)
                        {
                                TXB = 0;                    //send start bit
                                TDAT = TBUF;                //load data from TBUF to TDAT
                                TBIT = 9;           //initial send bit number (8 data bits + 1 stop bit)
                        }
                        else
                        {
                                TDAT >>= 1;                 //shift data to CY
                                if (--TBIT == 0)
                                {
                                        TXB = 1;
                                        TING = 0;           //stop send
                                        TEND = 1;           //set send completed flag
                                }
                                else
                                {
                                        TXB = CY;           //write CY to TX port
                                }
                        }
                }
        }
}

//-----------------------------------------
//initial UART module variable

void UART_INIT()
{
        TING = 0;
        RING = 0;
        TEND = 1;
        REND = 0;
        TCNT = 0;
        RCNT = 0;
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ------------------------------------------------------------------------------*/
/* --- STC15Fxx 系列 软件模拟串口举例---------------------------------------------------------------*/
/* --- 研发顾问QQ：800003751------------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8058芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz
;-----------------------------------------
;define baudrate const
;BAUD = 65536 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
;NOTE: (FOSC/3/BAUDRATE) must be greater then 75, (RECOMMEND GREATER THEN 100)

;BAUD   EQU     0F400H                  ; 1200bps @ 11.0592MHz
;BAUD   EQU     0FA00H                  ; 2400bps @ 11.0592MHz
;BAUD   EQU     0FD00H                  ; 4800bps @ 11.0592MHz
;BAUD   EQU     0FE80H                  ; 9600bps @ 11.0592MHz
;BAUD   EQU     0FF40H                  ;19200bps @ 11.0592MHz
;BAUD   EQU     0FFA0H                  ;38400bps @ 11.0592MHz
;BAUD   EQU     0FFC0H                  ;57600bps @ 11.0592MHz

;BAUD   EQU     0EC00H                  ; 1200bps @ 18.432MHz
;BAUD   EQU     0F600H                  ; 2400bps @ 18.432MHz
;BAUD   EQU     0FB00H                  ; 4800bps @ 18.432MHz
;BAUD   EQU     0FD80H                  ; 9600bps @ 18.432MHz
;BAUD   EQU     0FEC0H                  ;19200bps @ 18.432MHz
;BAUD   EQU     0FF60H                  ;38400bps @ 18.432MHz
BAUD    EQU     0FF95H                  ;57600bps @ 18.432MHz

;BAUD    EQU     0E800H                 ; 1200bps @ 22.1184MHz
;BAUD    EQU     0F400H                 ; 2400bps @ 22.1184MHz
;BAUD    EQU     0FA00H                 ; 4800bps @ 22.1184MHz
;BAUD    EQU     0FD00H                 ; 9600bps @ 22.1184MHz
;BAUD    EQU     0FE80H                 ;19200bps @ 22.1184MHz
;BAUD    EQU     0FF40H                 ;38400bps @ 22.1184MHz
;BAUD    EQU     0FF80H                 ;57600bps @ 22.1184MHz
```

STC MCU Limited

```
;-----------------------------------------
;define UART TX/RX port

RXB     BIT     P3.0
TXB     BIT     P3.1

;-----------------------------------------
;define  SFR

AUXR    DATA    8EH

;-----------------------------------------
;define UART module variable

TBUF    DATA    08H             ;(R0) ready send data buffer   (USER WRITE ONLY)
RBUF    DATA    09H             ;(R1) received data buffer     (UAER READ ONLY)
TDAT    DATA    0AH             ;(R2) sending data buffer      (RESERVED FOR UART MODULE)
RDAT    DATA    0BH             ;(R3) receiving data buffer    (RESERVED FOR UART MODULE)
TCNT    DATA    0CH             ;(R4) send baudrate counter    (RESERVED FOR UART MODULE)
RCNT    DATA    0DH             ;(R5) receive baudrate counter (RESERVED FOR UART MODULE)
TBIT    DATA    0EH             ;(R6) send bit counter         (RESERVED FOR UART MODULE)
RBIT    DATA    0FH             ;(R7) receive bit counter      (RESERVED FOR UART MODULE)

TING    BIT     20H.0           ; sending flag   (USER WRITE "1" TO TRIGGER SEND DATA,
                                ;                 CLEAR BY MODULE)
RING    BIT     20H.1           ; receiving flag (RESERVED FOR UART MODULE)
TEND    BIT     20H.2           ; sent flag      (SET BY MODULE AND SHOULD USER CLEAR)
REND    BIT     20H.3           ; received flag  (SET BY MODULE AND SHOULD USER CLEAR)

RPTR    DATA    21H             ;circular queue read pointer
WPTR    DATA    22H             ;circular queue write pointer
BUFFER  DATA    23H             ;circular queue buffer (16 bytes)

;-----------------------------------------
        ORG     0000H
        LJMP    RESET

;-----------------------------------------
;Timer0 interrupt routine for UART

        ORG     000BH

        PUSH    ACC             ;4 save ACC
        PUSH    PSW             ;4 save PSW
```

STC MCU Limited

```
        MOV     PSW,    #08H            ;3 using register group 1
L_UARTSTART:
;--------------------
        JB      RING,   L_RING          ;4 judge whether receiving
        JB      RXB,    L_REND          ;  check start signal
L_RSTART:
        SETB    RING                    ;  set start receive flag
        MOV     R5,     #4              ;  initial receive baudrate counter
        MOV     R7,     #9              ;  initial receive bit number (8 data bits + 1 stop bit)
        SJMP    L_REND                  ;  end this time slice
L_RING:
        DJNZ    R5,     L_REND          ;4 judge whether sending
        MOV     R5,     #3              ;2 reset send baudrate counter
L_RBIT:
        MOV     C,      RXB             ;3 read RX port data
        MOV     A,      R3              ;1 and shift it to RX buffer
        RRC     A                       ;1
        MOV     R3,     A               ;2
        DJNZ    R7,     L_REND          ;4 judge whether the data have receive completed
L_RSTOP:
        RLC     A                       ;  shift out stop bit
        MOV     R1,     A               ;  save the data to RBUF
        CLR     RING                    ;  stop receive
        SETB    REND                    ;  set receive completed flag
L_REND:
;--------------------
L_TING:
        DJNZ    R4,     L_TEND          ;4 check send baudrate counter
        MOV     R4,     #3              ;2 reset it
        JNB     TING,   L_TEND          ;4 judge whether sending
        MOV     A,      R6              ;1 detect the sent bits
        JNZ     L_TBIT                  ;3 "0" means start bit not sent
L_TSTART:
        CLR     TXB                     ;  send start bit
        MOV     TDAT,   R0              ;  load data from TBUF to TDAT
        MOV     R6,     #9              ;  initial send bit number (8 data bits + 1 stop bit)
        JMP     L_TEND                  ;  end this time slice
L_TBIT:
        MOV     A,      R2              ;1 read data in TDAT
        SETB    C                       ;1 shift in stop bit
        RRC     A                       ;1 shift data to CY
        MOV     R2,     A               ;2 update TDAT
        MOV     TXB,    C               ;4 write CY to TX port
        DJNZ    R6,     L_TEND          ;4 judge whether the data have send completed
```

STC MCU Limited

```
L_TSTOP:
        CLR     TING                    ;  stop send
        SETB    TEND                    ;  set send completed flag
L_TEND:
;------------------
L_UARTEND:
        POP     PSW                     ;3 restore PSW
        POP     ACC                     ;3 restore ACC
        RETI                            ;4 (69)
;----------------------------------------
;initial UART module variable

UART_INIT:
        CLR     TING
        CLR     RING
        SETB    TEND
        CLR     REND
        CLR     A
        MOV     TCNT,   A
        MOV     RCNT,   A
        RET
;----------------------------------------
;main program entry

RESET:
        MOV     R0,     #7FH            ;clear RAM
        CLR     A
        MOV     @R0,    A
        DJNZ    R0,     $-1
        MOV     SP,     #7FH            ;initial SP
;--------------------
;system initial
        MOV     TMOD,   #00H            ;timer0 in 16-bit auto reload mode
        MOV     AUXR,   #80H            ;timer0 working at 1T mode
        MOV     TL0,    #LOW    BAUD    ;initial timer0 and
        MOV     TH0,    #HIGH   BAUD    ;set reload value
        SETB    TR0                     ;tiemr0 start running
        SETB    ET0                     ;enable timer0 interrupt
        SETB    PT0                     ;improve timer0 interrupt priority
        SETB    EA                      ;open global interrupt switch
        LCALL   UART_INIT
;----------------------------------------
MAIN:
        JNB     REND,   CHECKREND               ;if (REND)
```

STC MCU Limited

```
        CLR     REND                    ;{
        MOV     A,      RPTR            ;   REND = 0;
        INC     RPTR                    ;   BUFFER[RPTR++ & 0xf] = RBUF;
        ANL     A,      #0FH            ;}
        ADD     A,      #BUFFER         ;
        MOV     R0,     A               ;
        MOV     @R0,    RBUF            ;
CHECKREND:
        JNB     TEND,   MAIN            ;if (TEND)
        MOV     A,      RPTR            ;{
        XRL     A,      WPTR            ;   if (WPTR != REND)
        JZ      MAIN                    ;   {
        CLR     TEND                    ;       TEND = 0;
        MOV     A,      WPTR            ;       TBUF = BUFFER[WPTR++ & 0xf];
        INC     WPTR                    ;       TING = 1;
        ANL     A,      #0FH            ;   }
        ADD     A,      #BUFFER         ;}
        MOV     R0,     A               ;
        MOV     TBUF,   @R0             ;
        SETB    TING                    ;
        SJMP    MAIN

;----------------------------------------

        END
```

STC MCU Limited

8.17 用T2结合INT4模拟一个半双工串口的测试程序(C及汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/

/*************  功能说明  ***************
使用STC15系列的Timer2做的模拟串口. P3.0接收, P3.1发送, 半双工.

假定测试芯片的工作频率为22118400Hz. 时钟为5.5296MHZ ~ 35MHZ.

波特率高，则时钟也要选高, 优先使用 22.1184MHZ, 11.0592MHZ.

测试方法: 上位机发送数据, MCU收到数据后原样返回.

串口固定设置: 1位起始位, 8位数据位, 1位停止位, 波特率在范围如下.

1200 ~ 115200 bps @ 33.1776MHZ
 600 ~ 115200 bps @ 22.1184MHZ
 600 ~  76800 bps @ 18.4320MHZ
 300 ~  57600 bps @ 11.0592MHZ
 150 ~  19200 bps @  5.5296MHZ
*****************************************/
#include  <reg52.h>

#define   MAIN_Fosc       22118400UL          //定义主时钟
#define   UART3_Baudrate  115200UL            //定义波特率
#define   RX_Lenth        32                  //接收长度

#define   UART3_BitTime   (MAIN_Fosc / UART3_Baudrate)

typedef   unsigned char   u8;
typedef   unsigned int    u16;
typedef   unsigned long   u32;

sfr       IE2       =     0xAF;
sfr       AUXR      =     0x8E;
sfr       INT_CLKO  =     0x8F;
sfr       T2H       =     0xD6;
sfr       T2L       =     0xD7;
u8        Tx3_read;                   //发送读指针
u8        Rx3_write;                  //接收写指针
```

```
u8 idata   buf3[RX_Lenth];                  //接收缓冲

u16        RxTimeOut;
bit        B_RxOk;                          //接收结束标志

//====================== 模拟串口相关============================
sbit       P_RX3 = P3^0;                    //定义模拟串口接收IO
sbit       P_TX3 = P3^1;                    //定义模拟串口发送IO

u8         Tx3_DAT;                         // 发送移位变量, 用户不可见
u8         Rx3_DAT;                         // 接收移位变量, 用户不可见
u8         Tx3_BitCnt;                      // 发送数据的位计数器, 用户不可见
u8         Rx3_BitCnt;                      // 接收数据的位计数器, 用户不可见
u8         Rx3_BUF;                         // 接收到的字节, 用户读取
u8         Tx3_BUF;                         // 要发送的字节, 用户写入
bit        Rx3_Ring;                        // 正在接收标志, 低层程序使用, 用户程序不可见
bit        Tx3_Ting;                        // 正在发送标志, 用户置1请求发送, 底层发送完成清0
bit        RX3_End;                         // 接收到一个字节, 用户查询 并清0
//==============================================================
void       UART_Init(void);

/******************** 主函数 **************************/
void main(void)
{
        UART_Init();                                    //PCA初始化
        EA = 1;

        while (1)                                       //user's function
        {
                if (RX3_End)                            // 检测是否收到一个字节
                {
                        RX3_End = 0;                    // 清除标志
                        buf3[Rx3_write] = Rx3_BUF;      // 写入缓冲
                        if(++Rx3_write >= RX_Lenth)     Rx3_write = 0;
                                                        // 指向下一个位置, 溢出检测
                        RxTimeOut = 1000;               //装载超时时间
                }
                if(RxTimeOut != 0)                      // 超时时间是否非0?
                {
                        if(--RxTimeOut == 0)            // (超时时间  - 1) == 0?
                        {
                                B_RxOk = 1;
                                AUXR &=  ~(1<<4);       //Timer2 停止运行
                                INT_CLKO &= ~(1 << 6);  //禁止INT4中断
                                T2H = (65536 - UART3_BitTime) / 256;        //数据位
                                T2L = (65536 - UART3_BitTime) % 256;        //数据位
                                AUXR |=  (1<<4);  //Timer2 开始运行
                        }
                }
```

STC MCU Limited

```
                if(B_RxOk)                                  // 检测是否接收OK?
                {
                        if (!Tx3_Ting)                      // 检测是否发送空闲
                        {
                                if (Tx3_read != Rx3_write)  // 检测是否收到过字符
                                {
                                        Tx3_BUF = buf3[Tx3_read]; // 从缓冲读一个字符发送
                                        Tx3_Ting = 1;             // 设置发送标志
                                        if(++Tx3_read >= RX_Lenth)      Tx3_read = 0;
                                                                  // 指向下一个位置, 溢出检测
                                }
                                else
                                {
                                        B_RxOk = 0;
                                        AUXR &=  ~(1<<4);         //Timer2 停止运行
                                        INT_CLKO |=  (1 << 6);    //允许INT4中断
                                }
                        }
                }
        }
}

//========================================================================
// 函数: void        UART_Init(void)
// 描述: UART初始化程序.
// 参数: none
// 返回: none.
// 版本: V1.0, 2013-11-22
//========================================================================
void    UART_Init(void)
{
        Tx3_read  = 0;
        Rx3_write = 0;
        Tx3_Ting  = 0;
        Rx3_Ring  = 0;
        RX3_End   = 0;
        Tx3_BitCnt = 0;
        RxTimeOut = 0;
        B_RxOk = 0;

        AUXR &=  ~(1<<4);                                   // Timer2 停止运行
        T2H = (65536 - UART3_BitTime) / 256;                // 数据位
        T2L = (65536 - UART3_BitTime) % 256;                // 数据位
        INT_CLKO |=  (1 << 6);                              // 允许INT4中断
        IE2  |=  (1<<2);                                    // 允许Timer2中断
        AUXR |=  (1<<2);                                    // 1T
}
//========================================================================
//========================================================================
```

STC MCU Limited

```
// 函数: void        timer2_int (void) interrupt 12
// 描述: Timer2中断处理程序.
// 参数: None
// 返回: none.
// 版本: V1.0, 2012-11-22
//========================================================================
void    timer2_int (void) interrupt 12
{
        if(Rx3_Ring)                            //已收到起始位
        {
                if (--Rx3_BitCnt == 0)          //接收完一帧数据
                {
                        Rx3_Ring = 0;           //停止接收
                        Rx3_BUF = Rx3_DAT;      //存储数据到缓冲区
                        RX3_End = 1;
                        AUXR &=  ~(1<<4);       //Timer2 停止运行
                        INT_CLKO |=  (1 << 6);  //允许INT4中断
                }
                else
                {
                        Rx3_DAT >>= 1;          //把接收的单b数据 暂存到 RxShiftReg(接收缓冲)
                        if(P_RX3) Rx3_DAT |= 0x80;  //shift RX data to RX buffer
                }
        }

        if(Tx3_Ting)                            // 不发送, 退出
        {
                if(Tx3_BitCnt == 0)             //发送计数器为0 表明单字节发送还没开始
                {
                        P_TX3 = 0;              //发送开始位
                        Tx3_DAT = Tx3_BUF;      //把缓冲的数据放到发送的buff
                        Tx3_BitCnt = 9;         //发送数据位数 (8数据位+1停止位)
                }
                else                            //发送计数器为非0 正在发送数据
                {
                        if (--Tx3_BitCnt == 0)  //发送计数器减为0 表明单字节发送结束
                        {
                                P_TX3 = 1;      //送停止位数据
                                Tx3_Ting = 0;   //发送停止
                        }
                        else
                        {
                                Tx3_DAT >>= 1;  //把最低位送到 CY(益处标志位)
                                P_TX3 = CY;     //发送一个bit数据
                        }
                }
        }
}
```

STC MCU Limited

```
/********************** INT4中断函数 *************************/
void Ext_INT4 (void) interrupt 16
{
    AUXR &=  ~(1<<4);                                       //Timer2 停止运行
    T2H = (65536 - (UART3_BitTime / 2 + UART3_BitTime)) / 256;    //起始位 + 半个数据位
    T2L = (65536 - (UART3_BitTime / 2 + UART3_BitTime)) % 256;    //起始位 + 半个数据位
    AUXR |=  (1<<4);  //Timer2 开始运行
    Rx3_Ring = 1;                                           //标志已收到起始位
    Rx3_BitCnt = 9;                                 //初始化接收的数据位数(8个数据位+1个停止位)

    INT_CLKO &= ~(1 << 6);                          //禁止INT4中断
    T2H = (65536 - UART3_BitTime) / 256;            //数据位
    T2L = (65536 - UART3_BitTime) % 256;            //数据位
}
```

2. 汇编程序：

```
;/*------------------------------------------------------------------------------------------------*/
;/* --- STC MCU Limited ------------------------------------------------------------------------*/
;/* --- Mobile: (86)13922805190 ----------------------------------------------------------------*/
;/* --- Fax: 86-755-82905966 -------------------------------------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------------------------------------*/
;/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
;/*------------------------------------------------------------------------------------------------*/

;**************     功能说明        ***************
;使用STC15系列的Timer2做的模拟串口. P3.0接收, P3.1发送, 半双工.

;假定测试芯片的工作频率为22118400Hz. 时钟为5.5296MHZ ~ 35MHZ.

;波特率高，则时钟也要选高, 优先使用 22.1184MHZ, 11.0592MHZ.

;测试方法: 上位机发送数据, MCU收到数据后原样返回.

;串口固定设置: 1位起始位, 8位数据位, 1位停止位，波特率在范围如下.

;*******************************************

STACK_POIRTER EQU       0D0H    ;堆栈开始地址

;UART3_BitTime   EQU      9216    ; 1200bps @ 11.0592MHz
                                 ; UART3_BitTime = (MAIN_Fosc / Baudrate)
;UART3_BitTime   EQU      4608    ; 2400bps @ 11.0592MHz
;UART3_BitTime   EQU      2304    ; 4800bps @ 11.0592MHz
;UART3_BitTime   EQU      1152    ; 9600bps @ 11.0592MHz
;UART3_BitTime   EQU       576    ;19200bps @ 11.0592MHz
;UART3_BitTime   EQU       288    ;38400bps @ 11.0592MHz
;UART3_BitTime   EQU               192       ;57600bps @ 11.0592MHz
```

```
;UART3_BitTime   EQU     15360           ; 1200bps @ 18.432MHz
;UART3_BitTime   EQU     7680            ; 2400bps @ 18.432MHz
;UART3_BitTime   EQU     3840            ; 4800bps @ 18.432MHz
;UART3_BitTime   EQU     1920            ; 9600bps @ 18.432MHz
;UART3_BitTime   EQU     960             ;19200bps @ 18.432MHz
;UART3_BitTime   EQU     480             ;38400bps @ 18.432MHz
;UART3_BitTime   EQU     320             ;57600bps @ 18.432MHz

;UART3_BitTime   EQU     18432           ; 1200bps @ 22.1184MHz
;UART3_BitTime   EQU     9216            ; 2400bps @ 22.1184MHz
;UART3_BitTime   EQU     4608            ; 4800bps @ 22.1184MHz
;UART3_BitTime   EQU     2304            ; 9600bps @ 22.1184MHz
;UART3_BitTime   EQU     1152            ;19200bps @ 22.1184MHz
;UART3_BitTime   EQU     576             ;38400bps @ 22.1184MHz
;UART3_BitTime   EQU     384             ;57600bps @ 22.1184MHz
UART3_BitTime   EQU     192             ;115200bps @ 22.1184MHz

;UART3_BitTime   EQU     27648           ; 1200bps @ 33.1776MHz
;UART3_BitTime   EQU     13824           ; 2400bps @ 33.1776MHz
;UART3_BitTime   EQU     6912            ; 4800bps @ 33.1776MHz
;UART3_BitTime   EQU     3456            ; 9600bps @ 33.1776MHz
;UART3_BitTime   EQU     1728            ;19200bps @ 33.1776MHz
;UART3_BitTime   EQU     864             ;38400bps @ 33.1776MHz
;UART3_BitTime   EQU     576             ;57600bps @ 33.1776MHz
;UART3_BitTime   EQU     288             ;115200bps @ 33.1776MHz

IE2             DATA    0AFH
AUXR            DATA    08EH
INT_CLKO        DATA    08FH
T2H             DATA    0D6H
T2L             DATA    0D7H

;====================== 模拟串口相关 ============================
P_RX3           BIT     P3.0            ; 定义模拟串口接收IO
P_TX3           BIT     P3.1            ; 定义模拟串口发送IO

Rx3_Ring        BIT     20H.0           ; 正在接收标志, 低层程序使用, 用户程序不可见
Tx3_Ting        BIT     20H.1           ; 正在发送标志, 用户置1请求发送, 底层发送完成清0
RX3_End         BIT     20H.2           ; 接收到一个字节, 用户查询 并清0
B_RxOk          BIT     20H.3           ; 接收结束标志

Tx3_DAT         DATA    30H             ; 发送移位变量, 用户不可见
Rx3_DAT         DATA    31H             ; 接收移位变量, 用户不可见
Tx3_BitCnt      DATA    32H             ; 发送数据的位计数器, 用户不可见
Rx3_BitCnt      DATA    33H             ; 接收数据的位计数器, 用户不可见
Rx3_BUF         DATA    34H             ; 接收到的字节, 用户读取
Tx3_BUF         DATA    35H             ; 要发送的字节, 用户写入

;================================================================
```

STC MCU Limited

```
RxTimeOutH      DATA    36H             ;
RxTimeOutL      DATA    37H             ;

Tx3_read DATA   38H                     ; 发送读指针
Rx3_write       DATA    39H             ; 接收写指针

RX_Lenth        EQU     32              ; 接收长度
buf3            EQU     40H             ; 40H ~ 5FH     接收缓冲

;*******************************************************************
;*******************************************************************
        ORG     00H                     ;reset
        LJMP    F_Main

        ORG     63H                     ;12  Timer2 interrupt
        LJMP    F_Timer2_Interrupt

        ORG     83H                     ;16  INT4 interrupt
        LJMP    F_INT4_Interrupt

;******************** 主程序 **************************/
F_Main:

        MOV     SP, #STACK_POIRTER
        MOV     PSW, #0
        USING   0                       ;选择第0组R0~R7

;================= 用户初始化程序 ====================================
        LCALL   F_UART_Init             ;UART初始化
        SETB    EA

;=================== 主循环 ==================================
L_MainLoop:
        JNB     RX3_End, L_QuitRx3      ; 检测是否收到一个字节
        CLR     RX3_End                 ; 清除标志
        MOV     A, #buf3
        ADD     A, Rx3_write
        MOV     R0, A
        MOV     @R0, Rx3_BUF            ; 写入缓冲
        MOV     RxTimeOutH, #HIGH 1000  ; 装载超时时间
        MOV     RxTimeOutL, #LOW  1000  ;
        INC     Rx3_write               ; 指向下一个位置
        MOV     A, Rx3_write
        CLR     C
        SUBB    A, #RX_Lenth            ; 溢出检测
        JC      L_QuitRx3
        MOV     Rx3_write, #0
L_QuitRx3:
        MOV     A, RxTimeOutL
```

STC MCU Limited

```
        ORL     A, RxTimeOutH
        JZ      L_QuitTimeOut                   ; 超时时间是否非0?
        MOV     A, RxTimeOutL
        DEC     RxTimeOutL                      ; (超时时间  - 1) == 0?
        JNZ     $+4
        DEC     RxTimeOutH
        DEC     A
        ORL     A, RxTimeOutH
        JNZ     L_QuitTimeOut

        SETB    B_RxOk                          ; 超时, 标志接收完成
        ANL     AUXR, #NOT (1 SHL 4)            ; Timer2 停止运行
        ANL     INT_CLKO, #NOT (1 SHL 6)        ; 禁止INT4中断
        MOV     T2H, #HIGH (65536 - UART3_BitTime)  ;数据位
        MOV     T2L, #LOW (65536 - UART3_BitTime)   ;
        ORL     AUXR, #(1 SHL 4)                ; Timer2 开始运行
L_QuitTimeOut:
        JNB     B_RxOk, L_QuitTx3               ; 检测是否接收OK?
        JB      Tx3_Ting, L_QuitTx3             ; 检测是否发送空闲
        MOV     A, Tx3_read
        XRL     A, Rx3_write
        JZ      L_TxFinish                      ; 检测是否发送完毕

        MOV     A, #buf3
        ADD     A, Tx3_read
        MOV     R0, A
        MOV     Tx3_BUF, @R0                    ; 从缓冲读一个字符发送
        SETB    Tx3_Ting                        ; 设置发送标志
        INC     Tx3_read                        ; 指向下一个字符位置
        MOV     A, Tx3_read
        CLR     C
        SUBB    A, #RX_Lenth
        JC      L_QuitTx3                       ; 溢出检测
        MOV     Tx3_read, #0
        SJMP    L_QuitTx3
L_TxFinish:
        CLR     B_RxOk
        ANL     AUXR, #NOT (1 SHL 4)            ; Timer2 停止运行
        ORL     INT_CLKO, #(1 SHL 6)            ; 允许INT4中断

L_QuitTx3:
        LJMP    L_MainLoop

;================== 主程序结束 ================================
;==============================================================================
; 函数: F_UART_Init
; 描述: UART初始化程序.
; 参数: none
; 返回: none.
```

STC MCU Limited

```
; 版本: V1.0, 2013-11-22
;======================================================================
F_UART_Init:
        MOV     Tx3_read,  #0
        MOV     Rx3_write, #0
        CLR     Tx3_Ting
        CLR     RX3_End
        CLR     Rx3_Ring
        MOV     Tx3_BitCnt, #0

        MOV     RxTimeOutH, #0
        MOV     RxTimeOutL, #0
        CLR     B_RxOk

        ANL     AUXR, #NOT(1 SHL 4)                 ; Timer2 停止运行
        MOV     T2H, #HIGH (65536 - UART3_BitTime)  ; 数据位
        MOV     T2L, #LOW (65536 - UART3_BitTime)   ; 数据位
        ORL     INT_CLKO, #(1 SHL 6)                ; 允许INT4中断
        ORL     IE2, #(1 SHL 2)                     ; 允许Timer2中断
        ORL     AUXR, #(1 SHL 2)                    ; 1T模式
        RET
;======================================================================
;======================================================================
; 函数: void      F_Timer2_Interrupt
; 描述: Timer2中断处理程序.
; 参数: None
; 返回: none.
; 版本: V1.0, 2012-11-22
;======================================================================
F_Timer2_Interrupt:
        PUSH    PSW
        PUSH    ACC

        JNB     Rx3_Ring, L_QuitRx      ; 已收到起始位
        DJNZ    Rx3_BitCnt, L_RxBit     ; 接收完一帧数据

        CLR     Rx3_Ring                ; 停止接收
        MOV     Rx3_BUF, Rx3_DAT        ; 存储数据到缓冲区
        SETB    RX3_End                 ;
        ANL     AUXR, #NOT (1 SHL 4)    ; Timer2 停止运行
        ORL     INT_CLKO, #(1 SHL 6)    ; 允许INT4中断
        SJMP    L_QuitRx

L_RxBit:
        MOV     A, Rx3_DAT              ; 把接收的单b数据 暂存到 RxShiftReg(接收缓冲)
        MOV     C, P_RX3
        RRC     A
        MOV     Rx3_DAT, A
L_QuitRx:
```

STC MCU Limited

```
        JNB     Tx3_Ting, L_QuitTx              ; 不发送, 退出
        MOV     A, Tx3_BitCnt
        JNZ     L_TxData                        ; 发送计数器为0 表明单字节发送还没开始
        CLR     P_TX3                           ; 发送开始位
        MOV     Tx3_DAT, Tx3_BUF                ; 把缓冲的数据放到发送的buff
        MOV     Tx3_BitCnt, #9                  ; 发送数据位数 (8数据位+1停止位)
        SJMP    L_QuitTx
L_TxData:                                       ; 发送计数器为非0 正在发送数据
        DJNZ    Tx3_BitCnt, L_TxBit             ; 发送计数器减为0 表明单字节发送结束
        SETB    P_TX3                           ; 送停止位数据
        CLR     Tx3_Ting                        ; 发送停止
        SJMP    L_QuitTx
L_TxBit:
        MOV     A, Tx3_DAT                      ; 把最低位送到 CY(益处标志位)
        RRC     A
        MOV     P_TX3, C                        ; 发送一个bit数据
        MOV     Tx3_DAT, A
L_QuitTx:

        POP     ACC
        POP     PSW

        RETI

;====================== INT4中断函数 ==================================
F_INT4_Interrupt:
        PUSH    PSW
        PUSH    ACC

        ANL     AUXR, #NOT(1 SHL 4)     ;Timer2 停止运行
        MOV     T2H, #HIGH (65536 - (UART3_BitTime / 2 + UART3_BitTime))
                                                ; 起始位 + 半个数据位
        MOV     T2L, #LOW (65536 - (UART3_BitTime / 2 + UART3_BitTime))
                                                ; 起始位 + 半个数据位
        ORL     AUXR, #(1 SHL 4) ;Timer2 开始运行
        SETB    Rx3_Ring                        ; 标志已收到起始位
        MOV     Rx3_BitCnt, #9                  ; 初始化接收的数据位数(8个数据位+1个停止位)

        ANL     INT_CLKO,  #NOT(1 SHL 6);       ; 禁止INT4中断
        MOV     T2H, #HIGH (65536 - UART3_BitTime)      ; 数据位
        MOV     T2L, #LOW (65536 - UART3_BitTime)       ; 数据位

        POP     ACC
        POP     PSW
        RETI

        END
```

STC MCU Limited

8.18 利用两路CCP/PCA模拟一个全双工串口的程序(C及汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ------------------------------------------------------------------------------*/
/* --- Mobile: (86)13922805190 ----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------------------------------*/
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/

/*************    功能说明            ***************

使用STC15系列的PCA0和PCA1做的模拟串口. PCA0接收(P2.5), PCA1发送(P2.6).

假定测试芯片的工作频率为22118400Hz. 时钟为5.5296MHZ ~ 35MHZ.

波特率高，则时钟也要选高, 优先使用 22.1184MHZ, 11.0592MHZ.

测试方法: 上位机发送数据,MCU收到数据后原样返回.

串口固定设置: 1位起始位, 8位数据位, 1位停止位, 波特率在 600 ~ 57600 bps.

1200 ~ 57600 bps @ 33.1776MHZ
 600 ~ 57600 bps @ 22.1184MHZ
 600 ~ 38400 bps @ 18.4320MHZ
 300 ~ 28800 bps @ 11.0592MHZ
 150 ~ 14400 bps @  5.5296MHZ

*****************************************/
#include  <reg52.h>

#define MAIN_Fosc               22118400UL          //定义主时钟
#define   UART3_Baudrate        57600UL             //定义波特率
#define   RX_Lenth              16                  //接收长度

#define   PCA_P12_P11_P10_P37   (0<<4)
#define   PCA_P34_P35_P36_P37   (1<<4)
#define   PCA_P24_P25_P26_P27   (2<<4)
#define   PCA_Mode_Capture      0
#define   PCA_Mode_SoftTimer    0x48
#define   PCA_Clock_1T          (4<<1)
#define   PCA_Clock_2T          (1<<1)
#define   PCA_Clock_4T          (5<<1)
#define   PCA_Clock_6T          (6<<1)
#define   PCA_Clock_8T          (7<<1)
#define   PCA_Clock_12T         (0<<1)
```

```
#define   PCA_Clock_ECI          (3<<1)
#define   PCA_Rise_Active        (1<<5)
#define   PCA_Fall_Active        (1<<4)
#define   PCA_PWM_8bit           (0<<6)
#define   PCA_PWM_7bit           (1<<6)
#define   PCA_PWM_6bit           (2<<6)

#define   UART3_BitTime   (MAIN_Fosc / UART3_Baudrate)

#define   ENABLE          1
#define   DISABLE         0

typedef   unsigned char   u8;
typedef   unsigned int    u16;
typedef   unsigned long   u32;

sfr       AUXR1  =        0xA2;
sfr       CCON   =        0xD8;
sfr       CMOD   =        0xD9;
sfr       CCAPM0=         0xDA;       //PCA模块0的工作模式寄存器。
sfr       CCAPM  =        0xDB;       //PCA模块1的工作模式寄存器。
sfr       CCAPM2=         0xDC;       //PCA模块2的工作模式寄存器。

sfr       CL     =        0xE9;
sfr       CCAP0L =        0xEA;       //PCA模块0的捕捉/比较寄存器低8位。
sfr       CCAP1L =        0xEB;       //PCA模块1的捕捉/比较寄存器低8位。
sfr       CCAP2L =        0xEC;       //PCA模块2的捕捉/比较寄存器低8位。

sfr       CH     =        0xF9;
sfr       CCAP0H =        0xFA;       //PCA模块0的捕捉/比较寄存器高8位。
sfr       CCAP1H =        0xFB;       //PCA模块1的捕捉/比较寄存器高8位。
sfr       CCAP2H =        0xFC;       //PCA模块2的捕捉/比较寄存器高8位。

sbit      CCF0   =        CCON^0;     //PCA 模块0中断标志，由硬件置位，必须由软件清0。
sbit      CCF1   =        CCON^1;     //PCA 模块1中断标志，由硬件置位，必须由软件清0。
sbit      CCF2   =        CCON^2;     //PCA 模块2中断标志，由硬件置位，必须由软件清0。
sbit      CR     =        CCON^6;     //1: 允许PCA计数器计数，0: 禁止计数。
sbit      CF     =        CCON^7;     //PCA计数器溢出（CH，CL由FFFFH变为0000H）标志。
                                       //PCA计数器溢出后由硬件置位，必须由软件清0。
sbit      PPCA   =        IP^7;       //PCA 中断 优先级设定位
u16       CCAP0_tmp;
u16       CCAP1_tmp;
u8        Tx3_read;                   //发送读指针
u8        Rx3_write;                  //接收写指针
u8 idata  buf3[RX_Lenth];             //接收缓冲

//==================== 模拟串口相关==========================
sbit      P_RX3 = P2^5;               //定义模拟串口接收IO
sbit      P_TX3 = P2^6;               //定义模拟串口发送IO
```

STC MCU Limited

```
u8      Tx3_DAT;                    // 发送移位变量, 用户不可见
u8      Rx3_DAT;                    // 接收移位变量, 用户不可见
u8      Tx3_BitCnt;                 // 发送数据的位计数器, 用户不可见
u       Rx3_BitCnt;                 // 接收数据的位计数器, 用户不可见
u8      Rx3_BUF;                    // 接收到的字节, 用户读取
u8      Tx3_BUF;                    // 要发送的字节, 用户写入
bit     Rx3_Ring;                   // 正在接收标志, 低层程序使用, 用户程序不可见
bit     Tx3_Ting;                   // 正在发送标志, 用户置1请求发送, 底层发送完成清0
bit     RX3_End;                    // 接收到一个字节, 用户查询 并清0
//==============================================================
void    PCA_Init(void);
/******************** 主函数 **************************/
void main(void)
{
        PCA_Init();                                     //PCA初始化
        EA = 1;

        Tx3_read  = 0;
        Rx3_write = 0;
        Tx3_Ting  = 0;
        Rx3_Ring  = 0;
        RX3_End   = 0;
        Tx3_BitCnt = 0;
        while (1)                                       //user's function
        {
                if (RX3_End)                            // 检测是否收到一个字节
                {
                        RX3_End = 0;                    // 清除标志
                        buf3[Rx3_write] = Rx3_BUF;      // 写入缓冲
                        if(++Rx3_write >= RX_Lenth)     Rx3_write = 0;
                                                        // 指向下一个位置，溢出检测
                }
                if (!Tx3_Ting)                          // 检测是否发送空闲
                {
                        if (Tx3_read != Rx3_write)      // 检测是否收到过字符
                        {
                                Tx3_BUF = buf3[Tx3_read]; // 从缓冲读一个字符发送
                                Tx3_Ting = 1;           // 设置发送标志
                                if(++Tx3_read >= RX_Lenth)      Tx3_read = 0;
                                                        // 指向下一个位置，溢出检测
                        }
                }
        }
}
//==========================================================================
// 函数: void      PCA_Init(void)
// 描述: PCA初始化程序.
// 参数: none
// 返回: none.
```

STC MCU Limited

```
// 版本: V1.0, 2013-11-22
//========================================================================
void    PCA_Init(void)
{
    CR = 0;
    CCAPM0 = (PCA_Mode_Capture | PCA_Fall_Active | ENABLE);   //16位下降沿捕捉中断模式

    CCAPM1    = PCA_Mode_SoftTimer | ENABLE;
    CCAP1_tmp = UART3_BitTime;
    CCAP1L    = (u8)CCAP1_tmp;                  //将影射寄存器写入捕获寄存器，先写CCAP0L
    CCAP1H    = (u8)(CCAP1_tmp >> 8);           //后写CCAP0H

    CH = 0;
    CL = 0;
    AUXR1 = (AUXR1 & ~(3<<4)) | PCA_P24_P25_P26_P27;                    //切换IO口
    CMOD  = (CMOD  & ~(7<<1)) | PCA_Clock_1T;                           //选择时钟源
    PPCA  = 1;                                                          // 高优先级中断
    CR    = 1;                                                          // 运行PCA定时器
}
//========================================================================
//========================================================================
// 函数: void        PCA_Handler (void) interrupt 7
// 描述: PCA中断处理程序.
// 参数: None
// 返回: none.
// 版本: V1.0, 2012-11-22
//========================================================================
void    PCA_Handler (void) interrupt 7
{
    if(CCF0)                                                            //PCA模块0中断
    {
        CCF0 = 0;                                                       //清PCA模块0中断标志
        if(Rx3_Ring)                                                    //已收到起始位
        {
            if (--Rx3_BitCnt == 0)                                      //接收完一帧数据
            {
                Rx3_Ring = 0;                                           //停止接收
                Rx3_BUF = Rx3_DAT;                                      //存储数据到缓冲区
                RX3_End = 1;
                CCAPM0 = (PCA_Mode_Capture | PCA_Fall_Active | ENABLE);
                                                                        //16位下降沿捕捉中断模式
            }
            else
            {
                Rx3_DAT >>= 1;   //把接收的单b数据 暂存到 RxShiftReg(接收缓冲)
                if(P_RX3) Rx3_DAT |= 0x80;                              //shift RX data to RX buffer
                CCAP0_tmp += UART3_BitTime;                             //数据位时间
                CCAP0L = (u8)CCAP0_tmp;                                 //将影射寄存器写入捕获寄存
                                                                        //器，先写CCAP0L
```

STC MCU Limited

```
                        CCAP0H = (u8)(CCAP0_tmp >> 8);    //后写CCAP0H
                    }
                }
                else
                {
                    CCAP0_tmp = ((u16)CCAP0H << 8) + CCAP0L;          //读捕捉寄存器
                    CCAP0_tmp += (UART3_BitTime / 2 + UART3_BitTime);//起始位 + 半个数据位
                    CCAP0L = (u8)CCAP0_tmp;                   //将影射寄存器写入捕获
                                                              //寄存器，先写CCAP0L
                    CCAP0H = (u8)(CCAP0_tmp >> 8);            //后写CCAP0H
                    CCAPM0 = (PCA_Mode_SoftTimer | ENABLE);//16位软件定时中断模式
                    Rx3_Ring = 1;                             //标志已收到起始位
                    Rx3_BitCnt = 9;                           //初始化接收的数据位数(8个数
                                                              //据位+1个停止位)
                }
            }

            if(CCF1) //PCA模块1中断, 16位软件定时中断模式
            {
                CCF1 = 0;                                     //清PCA模块1中断标志
                CCAP1_tmp += UART3_BitTime;
                CCAP1L = (u8)CCAP1_tmp;           //将影射寄存器写入捕获寄存器，先写CCAP0L
                CCAP1H = (u8)(CCAP1_tmp >> 8);    //后写CCAP0H

                if(Tx3_Ting)                      // 不发送, 退出
                {
                    if(Tx3_BitCnt == 0)           //发送计数器为0 表明单字节发送还没开始
                    {
                        P_TX3 = 0;                //发送开始位
                        Tx3_DAT = Tx3_BUF;        //把缓冲的数据放到发送的buff
                        Tx3_BitCnt = 9;           //发送数据位数 (8数据位+1停止位)
                    }
                    else                          //发送计数器为非0 正在发送数据
                    {
                        if (--Tx3_BitCnt == 0)    //发送计数器减为0 表明单字节发送结束
                        {
                            P_TX3 = 1;            //送停止位数据
                            Tx3_Ting = 0;         //发送停止
                        }
                        else
                        {
                            Tx3_DAT >>= 1;        //把最低位送到 CY(益处标志位)
                            P_TX3 = CY;           //发送一个bit数据
                        }
                    }
                }
            }
}
```

STC MCU Limited

2. 汇编程序:

```
;/*---------------------------------------------------------------------------------------------*/
;/* --- STC MCU Limited ---------------------------------------------------------------------*/
;/* --- Mobile: (86)13922805190 -------------------------------------------------------------*/
;/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
;/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
;/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
;/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
;/*---------------------------------------------------------------------------------------------*/

;*************     功能说明              ***************
;使用STC15系列的PCA0和PCA1做的模拟串口. PCA0接收(P2.5), PCA1发送(P2.6).

;假定测试芯片的工作频率为22118400Hz. 时钟为5.5296MHZ ~ 35MHZ.

;波特率高，则时钟也要选高, 优先使用 22.1184MHZ, 11.0592MHZ.

;测试方法: 上位机发送数据,MCU收到数据后原样返回.

;串口固定设置: 1位起始位, 8位数据位, 1位停止位.

;*********************************************

STACK_POIRTER EQU     0D0H              ;堆栈开始地址

;UART3_BitTime   EQU     9216            ; 1200bps @ 11.0592MHz
                                        ;UART3_BitTime = (MAIN_Fosc / Baudrate)
;UART3_BitTime   EQU     4608            ; 2400bps @ 11.0592MHz
;UART3_BitTime   EQU     2304            ; 4800bps @ 11.0592MHz
;UART3_BitTime   EQU     1152            ; 9600bps @ 11.0592MHz
;UART3_BitTime   EQU      576            ;19200bps @ 11.0592MHz
;UART3_BitTime   EQU      288            ;38400bps @ 11.0592MHz

;UART3_BitTime   EQU     15360           ; 1200bps @ 18.432MHz
;UART3_BitTime   EQU     7680            ; 2400bps @ 18.432MHz
;UART3_BitTime   EQU     3840            ; 4800bps @ 18.432MHz
;UART3_BitTime   EQU     1920            ; 9600bps @ 18.432MHz
;UART3_BitTime   EQU     960             ;19200bps @ 18.432MHz
;UART3_BitTime   EQU     480             ;38400bps @ 18.432MHz
;UART3_BitTime   EQU     320             ;57600bps @ 18.432MHz

;UART3_BitTime   EQU     18432           ; 1200bps @ 22.1184MHz
;UART3_BitTime   EQU     9216            ; 2400bps @ 22.1184MHz
;UART3_BitTime   EQU     4608            ; 4800bps @ 22.1184MHz
;UART3_BitTime   EQU     2304            ; 9600bps @ 22.1184MHz
;UART3_BitTime   EQU     1152            ;19200bps @ 22.1184MHz
;UART3_BitTime   EQU     576             ;38400bps @ 22.1184MHz
UART3_BitTime   EQU     384             ;57600bps @ 22.1184MHz
;UART3_BitTime   EQU     27648           ; 1200bps @ 33.1776MHz
```

STC MCU Limited

```
;UART3_BitTime  EQU     13824           ; 2400bps @ 33.1776MHz
;UART3_BitTime  EQU     6912            ; 4800bps @ 33.1776MHz
;UART3_BitTime  EQU     3456            ; 9600bps @ 33.1776MHz
;UART3_BitTime  EQU     1728            ;19200bps @ 33.1776MHz
;UART3_BitTime  EQU     864             ;38400bps @ 33.1776MHz
;UART3_BitTime  EQU     576             ;57600bps @ 33.1776MHz
;UART3_BitTime  EQU     288             ;115200bps @ 33.1776MHz

PCA_P12_P11_P10_P37     EQU     (0 SHL 4)
PCA_P34_P35_P36_P37     EQU     (1 SHL 4)
PCA_P24_P25_P26_P27     EQU     (2 SHL 4)
PCA_Mode_Capture        EQU     0
PCA_Mode_SoftTimer      EQU     048H
PCA_Clock_1T            EQU     (4 SHL 1)
PCA_Clock_2T            EQU     (1 SHL 1)
PCA_Clock_4T            EQU     (5 SHL 1)
PCA_Clock_6T            EQU     (6 SHL 1)
PCA_Clock_8T            EQU     (7 SHL 1)
PCA_Clock_12T           EQU     (0 SHL 1)
PCA_Clock_ECI           EQU     (3 SHL 1)
PCA_Rise_Active         EQU     (1 SHL 5)
PCA_Fall_Active         EQU     (1 SHL 4)

ENABLE                  EQU     1

AUXR1           DATA    0xA2
CCON            DATA    0xD8
CMOD            DATA    0xD9
CCAPM0          DATA    0xDA            ; PCA模块0的工作模式寄存器。
CCAPM1          DATA    0xDB            ; PCA模块1的工作模式寄存器。
CCAPM2          DATA    0xDC            ; PCA模块2的工作模式寄存器。

CL              DATA    0xE9
CCAP0L          DATA    0xEA            ; PCA模块0的捕捉/比较寄存器低8位。
CCAP1L          DATA    0xEB            ; PCA模块1的捕捉/比较寄存器低8位。
CCAP2L          DATA    0xEC            ; PCA模块2的捕捉/比较寄存器低8位。

CH              DATA    0xF9
CCAP0H          DATA    0xFA            ; PCA模块0的捕捉/比较寄存器高8位。
CCAP1H          DATA    0xFB            ; PCA模块1的捕捉/比较寄存器高8位。
CCAP2H          DATA    0xFC            ; PCA模块2的捕捉/比较寄存器高8位。

CCF0    BIT     CCON.0                  ; PCA 模块0中断标志，由硬件置位，必须由软件清0。
CCF1    BIT     CCON.1                  ; PCA 模块1中断标志，由硬件置位，必须由软件清0。
CCF2    BIT     CCON.2                  ; PCA 模块2中断标志，由硬件置位，必须由软件清0。
CR      BIT     CCON.6                  ; 1: 允许PCA计数器计数，0: 禁止计数。
CF      BIT     CCON.7                  ; PCA计数器溢出（CH，CL由FFFFH变为0000H）标志。
                                        ;PCA计数器溢出后由硬件置位，必须由软件清0。
PPCA    BIT     IP.7                    ; PCA 中断 优先级设定位
```

STC MCU Limited

```
;====================== 模拟串口相关 ============================
P_RX3           BIT     P2.5            ; 定义模拟串口接收IO
P_TX3           BIT     P2.6            ; 定义模拟串口发送IO

Rx3_Ring        BIT     20H.0           ; 正在接收标志, 低层程序使用, 用户程序不可见
Tx3_TingBIT     20H.1                   ; 正在发送标志, 用户置1请求发送, 底层发送完成清0
RX3_End         BIT     20H.2           ; 接收到一个字节, 用户查询 并清0

Tx3_DAT         DATA    30H             ; 发送移位变量, 用户不可见
Rx3_DAT         DATA    31H             ; 接收移位变量, 用户不可见
Tx3_BitCnt      DATA    32H             ; 发送数据的位计数器, 用户不可见
Rx3_BitCnt      DATA    33H             ; 接收数据的位计数器, 用户不可见
Rx3_BUF         DATA    34H             ; 接收到的字节, 用户读取
Tx3_BUF         DATA    35H             ; 要发送的字节, 用户写入
;=================================================================
Tx3_read DATA   36H                     ; 发送读指针
Rx3_write       DATA    37H             ; 接收写指针

RX_Lenth        EQU     16              ; 接收长度
buf3            EQU     40H             ; 40H ~ 4FH       接收缓冲
;*****************************************************************
;*****************************************************************
        ORG     00H                     ;reset
        LJMP    F_Main

        ORG     3BH                     ;7  PCA interrupt
        LJMP    F_PCA_Interrupt

;******************** 主程序 **************************/
F_Main:

        MOV     SP, #STACK_POIRTER
        MOV     PSW, #0
        USING   0                       ;选择第0组R0~R7

;================= 用户初始化程序 ====================================
        LCALL   F_PCA_Init              ;PCA初始化
        SETB    EA

        MOV     Tx3_read,  #0
        MOV     Rx3_write, #0
        CLR     Tx3_Ting
        CLR     RX3_End
        CLR     Rx3_Ring
        MOV     Tx3_BitCnt, #0

;================== 主循环 ==================================
L_MainLoop:
        JNB             RX3_End, L_QuitRx3      ; 检测是否收到一个字节
        CLR             RX3_End                                 ; 清除标志
```

STC MCU Limited

```
        MOV     A, #buf3
        ADD     A, Rx3_write
        MOV     R0, A
        MOV     @R0, Rx3_BUF                ; 写入缓冲
        INC     Rx3_write                   ; 指向下一个位置
        MOV     A, Rx3_write
        CLR     C
        SUBB    A, #RX_Lenth                ; 溢出检测
        JC      L_QuitRx3
        MOV     Rx3_write, #0
L_QuitRx3:

        JB      Tx3_Ting, L_QuitTx3         ;检测是否发送空闲
        MOV     A, Tx3_read
        XRL     A, Rx3_write
        JZ      L_QuitTx3                   ;检测是否收到过字符

        MOV     A, #buf3
        ADD     A, Tx3_read
        MOV     R0, A
        MOV     Tx3_BUF, @R0                ; 从缓冲读一个字符发送
        SETB    Tx3_Ting                    ; 设置发送标志
        INC     Tx3_read                    ; 指向下一个字符位置
        MOV     A, Tx3_read
        CLR     C
        SUBB    A, #RX_Lenth
        JC      L_QuitTx3                   ; 溢出检测
        MOV     Tx3_read, #0
L_QuitTx3:

        SJMP    L_MainLoop

;=================== 主程序结束 ==================================
;================================================================================
; 函数: F_PCA_Init
; 描述: PCA初始化程序.
; 参数: none
; 返回: none.
; 版本: V1.0, 2013-11-22
;================================================================================
F_PCA_Init:
        CLR             CR
        MOV             CCAPM0, #(PCA_Mode_Capture OR PCA_Fall_Active OR ENABLE)
                                            ; 16位下降沿捕捉中断模式
        MOV             CCAPM1, #(PCA_Mode_SoftTimer OR ENABLE)
                                            ; 16位软件定时器, 中断模式
        MOV             CCAP1L, #LOW UART3_BitTime
                                            ; 将影射寄存器写入捕获寄存器，先写CCAP0L
        MOV             CCAP1H, #HIGH UART3_BitTime         ; 后写CCAP0H
```

STC MCU Limited

```
        MOV     CH, #0
        MOV     CL, #0
        MOV     A, AUXR1
        ANL     A, # NOT(3 SHL 4)
        ORL     A, #PCA_P24_P25_P26_P27         ;切换IO口
        MOV     AUXR1, A
        ANL     A, #NOT(7 SHL 1)
        ORL     A, #PCA_Clock_1T                ;选择时钟源
        MOV     CMOD, A
        SETB    PPCA                            ; 高优先级中断
        SETB    CR                              ; 运行PCA定时器
        RET
;========================================================================
;========================================================================
; 函数: F_PCA_Interrupt
; 描述: PCA中断处理程序.
; 参数: None
; 返回: none.
; 版本: V1.0, 2012-11-22
;========================================================================
F_PCA_Interrupt:
        PUSH    PSW
        PUSH    ACC
        ;=============== PCA模块0中断 ====================
        JNB     CCF0, L_QuitPCA0                ; PCA模块0中断
        CLR     CCF0                            ; 清PCA模块0中断标志

        JNB     Rx3_Ring, L_Rx3_Start           ; 已收到起始位
        DJNZ    Rx3_BitCnt, L_RxBit             ; 接收完一帧数据

        CLR     Rx3_Ring                        ; 停止接收
        MOV     Rx3_BUF, Rx3_DAT                ; 存储数据到缓冲区
        SETB    RX3_End                                 ;
        MOV     CCAPM0, #(PCA_Mode_Capture OR PCA_Fall_Active OR ENABLE)
                                                ; 16位下降沿捕捉中断模式
        SJMP    L_QuitPCA0

L_RxBit:
        MOV     A, Rx3_DAT                      ; 把接收的单b数据 暂存到 RxShiftReg(接收缓冲)
        MOV     C, P_RX3
        RRC     A
        MOV     Rx3_DAT, A
        MOV     A, CCAP0L                       ;
        ADD     A, #LOW UART3_BitTime           ; 数据位时间
        MOV     CCAP0L, A                       ; 将影射寄存器写入捕获寄存器，先写CCAP0L
        MOV     A, CCAP0H                       ; 数据位时间
        ADDC    A, #HIGH UART3_BitTime  ;
        MOV     CCAP0H, A                       ; 后写CCAP0H
        SJMP    L_QuitPCA0
```

STC MCU Limited

```
L_Rx3_Start:
        MOV     CCAPM0, #(PCA_Mode_SoftTimer OR ENABLE)         ; 16位软件定时中断模式
        MOV     A, CCAP0L                                       ; 数据位时间
        ADD     A, #LOW (UART3_BitTime / 2 + UART3_BitTime)     ;
        MOV     CCAP0L, A                       ; 将影射寄存器写入捕获寄存器，先写CCAP0L
        MOV     A, CCAP0H                       ; 数据位时间
        ADDC    A, #HIGH (UART3_BitTime / 2 + UART3_BitTime)    ;
        MOV     CCAP0H, A                       ; 后写CCAP0H
        SETB    Rx3_Ring                        ; 标志已收到起始位
        MOV     Rx3_BitCnt, #9                  ; 初始化接收的数据位数(8个数据位+1个停止位)
L_QuitPCA0:

        ;================ PCA模块1中断 ====================
        JNB     CCF1, L_QuitPCA1                ; PCA模块1中断, 16位软件定时中断模式
        CLR     CCF1                            ; 清PCA模块1中断标志
        MOV     A, CCAP1L                       ;
        ADD     A, #LOW UART3_BitTime           ; 数据位时间
        MOV     CCAP1L, A                       ; 将影射寄存器写入捕获寄存器，先写CCAP0L
        MOV     A, CCAP1H                       ;
        ADDC    A, #HIGH UART3_BitTime          ; 数据位时间
        MOV     CCAP1H, A                       ; 后写CCAP0H

        JNB     Tx3_Ting, L_QuitPCA1            ; 不发送, 退出
        MOV     A, Tx3_BitCnt
        JNZ     L_TxData                        ; 发送计数器为0 表明单字节发送还没开始
        CLR     P_TX3                           ; 发送开始位
        MOV     Tx3_DAT, Tx3_BUF                ; 把缓冲的数据放到发送的buff
        MOV     Tx3_BitCnt, #9                  ; 发送数据位数 (8数据位+1停止位)
        SJMP    L_QuitPCA1
L_TxData:                                       ; 发送计数器为非0 正在发送数据
        DJNZ    Tx3_BitCnt, L_TxBit             ; 发送计数器减为0 表明单字节发送结束
        SETB    P_TX3                           ; 送停止位数据
        CLR     Tx3_Ting                        ; 发送停止
        SJMP    L_QuitPCA1
L_TxBit:
        MOV     A, Tx3_DAT                      ; 把最低位送到 CY(益处标志位)
        RRC     A
        MOV     P_TX3, C                        ; 发送一个bit数据
        MOV     Tx3_DAT, A
L_QuitPCA1:

        POP     ACC
        POP     PSW

        RETI

        END
```

STC MCU Limited

第9章 STC15系列单片机EEPROM的应用

STC15系列单片机内部集成了大容量的EEPROM，其与程序空间是分开的。利用ISP/IAP技术可将内部Data Flash当EEPROM，擦写次数在10万次以上。EEPROM可分为若干个扇区，每个扇区包含512字节。使用时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的。

EEPROM可用于保存一些需要在应用过程中修改并且掉电不丢失的参数数据。在用户程序中，可以对EEPROM进行字节读/字节编程/扇区擦除操作。在工作电压Vcc偏低时，建议不要进行EEPROM/IAP操作。

9.1 IAP及EEPROM新增特殊功能寄存器介绍

符号	描述	地址	位地址及符号 MSB LSB								复位值
IAP_DATA	ISP/IAP Flash Data Register	C2H									1111 1111B
IAP_ADDRH	ISP/IAP Flash Address High	C3H									0000 0000B
IAP_ADDRL	ISP/IAP Flash Address Low	C4H									0000 0000B
IAP_CMD	ISP/IAP Flash Command Register	C5H	-	-	-	-	-	-	MS1	MS0	xxxx x000B
IAP_TRIG	ISP/IAP Flash Command Trigger	C6H									xxxx xxxxB
IAP_CONTR	ISP/IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0	0000 x000B
PCON	Power Control	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B

STC MCU Limited

1. ISP/IAP数据寄存器IAP_DATA

IAP_DATA：ISP/IAP 操作时的数据寄存器。
ISP/IAP 从Flash读出的数据放在此处，向Flash写的数据也需放在此处

2. ISP/IAP地址寄存器IAP_ADDRH和IAP_ADDRL

IAP_ADDRH ：ISP/IAP 操作时的地址寄存器高八位。
IAP_ADDRL ：ISP/IAP 操作时的地址寄存器低八位。

3. ISP/IAP命令寄存器IAP_CMD

ISP/IAP命令寄存器IAP_CMD格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	name	-	-	-	-	-	-	MS1	MS0

MS1	MS0	命令 / 操作　模式选择
0	0	Standby　待机模式，无ISP操作
0	1	从用户的应用程序区对"Data Flash/EEPROM区"进行字节读
1	0	从用户的应用程序区对"Data Flash/EEPROM区"进行字节编程
1	1	从用户的应用程序区对"Data Flash/EEPROM区"进行扇区擦除

程序在用户应用程序区时，仅可以对数据Flash区(EEPROM)进行字节读/字节编程/扇区擦除
IAP15系列除外，IAP15系列可在用户应用程序区修改用户应用程序区。
特别声明：EEPROM也可以用MOVC指令读(MOVC访问的是程序存储器)，但起始地址不再是0000H,而是程序存储空间结束地址的下一个地址。

4. ISP/IAP命令触发寄存器IAP_TRIG

IAP_TRIG: ISP/IAP操作时的命令触发寄存器。
在IAPEN(IAP_CONTR.7) = 1 时,对IAP_TRIG先写入5Ah,再写入A5h,ISP/IAP命令才会生效。

ISP/IAP操作完成后，IAP地址高八位寄存器IAP_ADDRH、IAP地址低八位寄存器IAP_ADDRL和IAP命令寄存器IAP_CMD的内容不变。如果接下来要对下一个地址的数据进行ISP/IAP操作，需手动将该地址的高8位和低8位分别写入IAP_ADDRH和IAP_ADDRL寄存器。

每次IAP操作时，都要对IAP_TRIG先写入5AH，再写入A5H，ISP/IAP命令才会生效。

在每次触发前，需重新送字节读/字节编程/扇区擦除命令，在命令不改变时,不需重新送命令

5. ISP/IAP命令寄存器IAP_CONTR

ISP/IAP控制寄存器IAP_CONTR格式如下：

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	name	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT2	WT0

IAPEN: ISP/IAP功能允许位。0：禁止IAP读/写/擦除Data Flash/EEPROM
1：允许IAP读/写/擦除Data Flash/EEPROM

SWBS：软件选择复位后从用户应用程序区启动(送0)，还是从系统ISP监控程序区启动(送1)。要与SWRST直接配合才可以实现

SWRST：0：不操作； 1：软件控制产生复位，单片机自动复位。

CMD_FAIL：如果IAP地址(由IAP地址寄存器IAP_ADDRH和IAP_ADDRL的值决定)指向了非法地址或无效地址，且送了ISP/IAP命令,并对IAP_TRIG送5Ah/A5h触发失败，则CMD_FAIL为1,需由软件清零。

```
;在用户应用程序区(AP 区)软件复位并从用户应用程序区(AP 区)开始执行程序
MOV IAP_CONTR, #00100000B ;SWBS = 0(选择AP 区), SWRST = 1(软复位)
;在用户应用程序区(AP 区)软件复位并从系统ISP 监控程序区开始执行程序
MOV IAP_CONTR, #01100000B ;SWBS = 1(选择ISP 区), SWRST = 1(软复位)
;在系统ISP 监控程序区软件复位并从用户应用程序区(AP 区)开始执行程序
MOV IAP_CONTR, #00100000B ;SWBS = 0(选择AP 区), SWRST = 1(软复位)
;在系统ISP 监控程序区软件复位并从系统ISP 监控程序区开始执行程序
MOV IAP_CONTR, #01100000B ;SWBS = 1(选择ISP 区), SWRST = 1(软复位)
```

设置等待时间			CPU等待时间(多少个CPU工作时钟)			
WT2	WT1	WT0	Read/读 (2个时钟)	Program/编程 (=55us)	Sector Erase 扇区擦除 (=21ms)	Recommended System Clock 跟等待参数对应的推荐系统时钟
1	1	1	2个时钟	55个时钟	21012个时钟	≤ 1MHz
1	1	0	2个时钟	110个时钟	42024个时钟	≤ 2MHz
1	0	1	2个时钟	165个时钟	63036个时钟	≤ 3MHz
1	0	0	2个时钟	330个时钟	126072个时钟	≤ 6MHz
0	1	1	2个时钟	660个时钟	252144个时钟	≤ 12MHz
0	1	0	2个时钟	1100个时钟	420240个时钟	≤ 20MHz
0	0	1	2个时钟	1320个时钟	504288个时钟	≤ 24MHz
0	0	0	2个时钟	1760个时钟	672384个时钟	≤ 30MHz

6. 工作电压过低判断，此时不要进行EEPROM/IAP操作

PCON：电源控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	name	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

LVDF: 低压检测标志位,当工作电压Vcc低于低压检测门槛电压时，该位置1。该位要由软件清0

当低压检测电路发现工作电压Vcc偏低时，不要进行EEPROM/IAP操作。

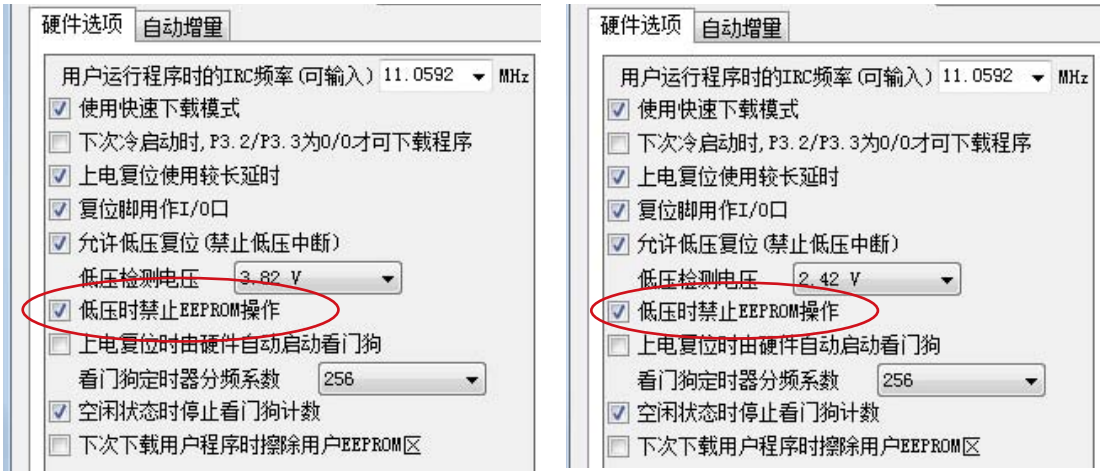
5V单片机的低压检测门槛电压：

-40 ℃	25 ℃	85 ℃
4.74	4.64	4.60
4.41	4.32	4.27
4.14	4.05	4.00
3.90	3.82	3.77
3.69	3.61	3.56
3.51	3.43	3.38
3.36	3.28	3.23
3.21	3.14	3.09

3.3V单片机的低压检测门槛电压：

-40 ℃	25 ℃	85 ℃
3.11	3.08	3.09
2.85	2.82	2.83
2.63	2.61	2.61
2.44	2.42	2.43
2.29	2.26	2.26
2.14	2.12	2.12
2.01	2.00	2.00
1.90	1.89	1.89

STC MCU Limited



建议在电压偏低时，不要操作EEPROM/IAP,烧录时直接选择“低压禁止EEPROM操作”

9.2 STC15系列单片机EEPROM空间大小及地址

9.2.1 STC15W4K60S4系列单片机EEPROM空间大小及地址

STC15W4K60S4系列单片机内部EEPROM选型一览表							STC15W4K60S4系列单片机内部EEPROM还可以用MOVC指令读，但此时首地址不再是0000H，而是程序存储空间结束地址的下一个地址 512字节为一个扇区
型号	EEPROM字节数	扇区数	用IAP字节读时EEPROM起始扇区首地址	用IAP字节读时EEPROM结束扇区末尾地址	用MOVC指令读时EEPROM起始扇区首地址	用MOVC指令读时EEPROM结束扇区末尾地址	
STC15W4K08S4	53K	106	0000h	D3FFh	2000h	F3FFh	
STC15W4K16S4	45K	90	0000h	B3FFh	4000h	F3FFh	
STC15W4K24S4	37K	74	0000h	93FFh	6000h	F3FFh	
STC15W4K32S4	29K	58	0000h	73FFh	8000h	F3FFh	
STC15W4K40S4	21K	42	0000h	53FFh	A000h	F3FFh	
STC15W4K48S4	13K	26	0000h	33FFh	C000h	F3FFh	
STC15W4K56S4	5K	10	0000h	13FFh	E000h	F3FFh	
STC15W4K60S4	1K	2	0000h	03FFh	F000h	F3FFh	
以下系列特殊，可在用户程序区直接修改程序，所有Flash空间均可作EEPROM修改							
IAP15W4K61S4	–	122	0000h	F3FFh			没有专门的EEPROM,但可在用户程序区修改用户程序，使用时不要将自己的有效程序擦除。

STC MCU Limited

9.2.2 STC15F2K60S2及STC15L2K60S2系列EEPROM空间大小及地址

STC15F2K60S2系列单片机内部EEPROM选型一览表 STC15L2K60S2系列单片机内部EEPROM选型一览表							STC15F2K60S2及STC15L2K60S2系列单片机内部EEPROM还可以用MOVC指令读，但此时首地址不再是0000H，而是程序存储空间结束地址的下一个地址 512字节为一个扇区
型号	EEPROM字节数	扇区数	用IAP字节读时EEPROM起始扇区首地址	用IAP字节读时EEPROM结束扇区末尾地址	用MOVC指令读时EEPROM起始扇区首地址	用MOVC指令读时EEPROM结束扇区末尾地址	
STC15F2K08S2 STC15L2K08S2	53K	106	0000h	D3FFh	2000h	F3FFh	
STC15F2K16S2 STC15L2K16S2	45K	90	0000h	B3FFh	4000h	F3FFh	
STC15F2K24S2 STC15L2K24S2	37K	74	0000h	93FFh	6000h	F3FFh	
STC15F2K32S2 STC15L2K32S2	29K	58	0000h	73FFh	8000h	F3FFh	
STC15F2K40S2 STC15L2K40S2	21K	42	0000h	53FFh	A000h	F3FFh	
STC15F2K48S2 STC15L2K48S2	13K	26	0000h	33FFh	C000h	F3FFh	
STC15F2K56S2 STC15L2K56S2	5K	10	0000h	13FFh	E000h	F3FFh	
STC15F2K60S2 STC15L2K60S2	1K	2	0000h	03FFh	F000h	F3FFh	
以下系列特殊，可在用户程序区直接修改程序，所有Flash空间均可作EEPROM修改							
IAP15F2K61S2 IAP15L2K61S2	–	122	0000h	F3FFh			没有专门的EEPROM，但可在用户程序区修改用户程序，使用时不要将自己的有效程序擦除。

STC MCU Limited

9.2.3 STC15W1K16S系列单片机EEPROM空间大小及地址

STC15W1K16S系列单片机内部EEPROM选型一览表							
型号	EEPROM字节数	扇区数	用IAP字节读时EEPROM起始扇区首地址	用IAP字节读时EEPROM结束扇区末尾地址	用MOVC指令读时EEPROM起始扇区首地址	用MOVC指令读时EEPROM结束扇区末尾地址	STC15W1K16S系列单片机内部EEPROM不可以用MOVC指令读 512字节为一个扇区
STC15W1K16S	13K	26	0000h	33FFh			
STC15W1K24S	5K	10	0000h	13FFh			
以下系列特殊，可在用户程序区直接修改程序，所有Flash空间均可作EEPROM修改							
IAP15W1K29S	–	58	0000h	73FFh			没有专门的EEPROM,但可在用户程序区修改用户程序，使用时不要将自己的有效程序擦除。

STC15W1K16S系列中以STC开头的单片机不能用MOVC读EEPROM。

STC MCU Limited

9.2.4 STC15W408S系列单片机EEPROM空间大小及地址

STC15W408S系列单片机内部EEPROM选型一览表							
型号	EEPROM字节数	扇区数	用IAP字节读时EEPROM起始扇区首地址	用IAP字节读时EEPROM结束扇区末尾地址	用MOVC指令读时EEPROM起始扇区首地址	用MOVC指令读时EEPROM结束扇区末尾地址	STC15W408S系列单片机内部EEPROM不可以用MOVC指令读 512字节为一个扇区
STC15W404S	9K	18	0000h	23FFh			
STC15W408S	5K	10	0000h	13FFh			
STC15W410S	3K	6	0000h	0BFFh			
以下系列特殊，可在用户程序区直接修改程序，所有Flash空间均可作EEPROM修改							
IAP15W413S	–	26	0000h	33FFh			没有专门的EEPROM,但可在用户程序区修改用户程序，使用时不要将自己的有效程序擦除。

STC15W408S系列中以STC开头的单片机不能用MOVC读EEPROM。

9.2.5 STC15W408AS系列单片机EEPROM空间大小及地址

STC15W408AS系列单片机内部EEPROM选型一览表							STC15W408AS系列单片机内部EEPROM还可以用MOVC指令读，但此时首地址不再是0000H，而是程序存储空间结束地址的下一个地址 512字节为一个扇区
型号	EEPROM字节数	扇区数	用IAP字节读时EEPROM起始扇区首地址	用IAP字节读时EEPROM结束扇区末尾地址	用MOVC指令读时EEPROM起始扇区首地址	用MOVC指令读时EEPROM结束扇区末尾地址	
STC15W404AS	9K	18	0000h	23FFh	1000h	33FFh	
STC15W408AS	5K	10	0000h	13FFh	2000h	33FFh	
STC15W410AD	3K	6	0000h	0BFFh	2800h	33FFh	
STC15W412AD	1K	2	0000h	03FFh	3000h	33FFh	
以下系列特殊，可在用户程序区直接修改程序，所有Flash空间均可作EEPROM修改							
IAP15W413AS	-	26	0000h	33FFh			没有专门的EEPROM,但可在用户程序区修改用户程序，使用时不要将自己的有效程序擦除。

STC MCU Limited

9.2.6 STC15W201S系列单片机EEPROM空间大小及地址

STC15W201S系列单片机内部EEPROM选型一览表							STC15W201S系列单片机内部EEPROM不可以用MOVC指令读 512字节为一个扇区
型号	EEPROM字节数	扇区数	用IAP字节读时EEPROM起始扇区首地址	用IAP字节读时EEPROM结束扇区末尾地址	用MOVC指令读时EEPROM起始扇区首地址	用MOVC指令读时EEPROM结束扇区末尾地址	
STC15W201S	4K	8	0000h	0FFFh			
STC15W202S	3K	6	0000h	0BFFh			
STC15W203S	2K	4	0000h	07FFh			
STC15W204S	1K	2	0000h	03FFh			
以下系列特殊，可在用户程序区直接修改程序，所有Flash空间均可作EEPROM修改							
IAP15W205S	-	10	0000h	13FFh			没有专门的EEPROM,但可在用户程序区修改用户程序，使用时不要将自己的有效程序擦除。

STC15W201S系列中以STC开头的单片机不能用MOVC读EEPROM。

9.2.7 STC15F408AD及STC15L408AD系列EEPROM空间大小及地址

STC15F408AD系列单片机内部EEPROM选型一览表 STC15L408AD系列单片机内部EEPROM选型一览表							STC15F408AD及STC15L408AD系列单片机内部EEPROM还可以用MOVC指令读，但此时首地址不再是0000H，而是程序存储空间结束地址的下一个地址 512字节为一个扇区
型号	EEPROM字节数	扇区数	用IAP字节读时EEPROM起始扇区首地址	用IAP字节读时EEPROM结束扇区末尾地址	用MOVC指令读时EEPROM起始扇区首地址	用MOVC指令读时EEPROM结束扇区末尾地址	
STC15F408AD STC15L408AD	5K	10	0000h	13FFh	2000h	33FFh	
以下系列特殊，可在用户程序区直接修改程序，所有Flash空间均可作EEPROM修改							
IAP15F413AD IAP15L413AD	–	26	0000h	33FFh			没有专门的EEPROM,但可在用户程序区修改用户程序，使用时不要将自己的有效程序擦除。

9.2.8 STC15F101W及STC15L101W系列EEPROM空间大小及地址

STC15F101W系列单片机内部EEPROM选型一览表 STC15L101W系列单片机内部EEPROM选型一览表							STC15F101W及STC15L101W系列单片机内部EEPROM还可以用MOVC指令读，但此时首地址不再是0000H，而是程序存储空间结束地址的下一个地址 512字节为一个扇区
型号	EEPROM字节数	扇区数	用IAP字节读时EEPROM起始扇区首地址	用IAP字节读时EEPROM结束扇区末尾地址	用MOVC指令读时EEPROM起始扇区首地址	用MOVC指令读时EEPROM结束扇区末尾地址	
STC15F101W STC15L101W	4K	8	0000h	0FFFh	0400h	13FFh	
STC15F102W STC15L102W	3K	6	0000h	0BFFh	0800h	13FFh	
STC15F103W STC15L103W	2K	4	0000h	07FFh	0C00h	13FFh	
STC15F104W STC15L104W	1K	2	0000h	03FFh	1000h	13FFh	
以下系列特殊，可在用户程序区直接修改程序，所有Flash空间均可作EEPROM修改							
IAP15F105W IAP15L105W	–	10	0000h	13FFh			没有专门的EEPROM,但可在用户程序区修改用户程序，使用时不要将自己的有效程序擦除。

STC MCU Limited

STC15单片机的内部EEPROM地址表

第一扇区		第二扇区		第三扇区		第四扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
0000h	1FFh	200h	3FFh	400h	5FFh	600h	7FFh
第五扇区		第六扇区		第七扇区		第八扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
800h	9FFh	A00h	BFFh	C00h	DFFh	E00h	FFFh
第九扇区		第十扇区		第十一扇区		第十二扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
1000h	11FFh	1200h	13FFh	1400h	15FFh	1600h	17FFh
第十三扇区		第十四扇区		第十五扇区		第十六扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
1800h	19FFh	1A00h	1BFFh	1C00h	1DFFh	1E00h	1FFFh
第十七扇区		第十八扇区		第十九扇区		第二十扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
2000h	21FFh	2200h	23FFh	2400h	25FFh	2600h	27FFh
第二十一扇区		第二十二扇区		第二十三扇区		第二十四扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
2800h	29FFh	2A00h	2BFFh	2C00h	2DFFh	2E00h	2FFFH
第二十五扇区		第二十六扇区		第二十七扇区		第二十八扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
3000h	31FFh	3200h	33FFh	3400h	35FFh	3600h	37FFH
第二十九扇区		第三十扇区		第三十一扇区		第三十二扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
3800h	39FFh	3A00h	3BFFh	3C00h	3DFFh	3E00h	3FFFH
第三十三扇区		第三十四扇区		第三十五扇区		第三十六扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
4000h	41FFh	4200h	43FFh	4400h	45FFh	4600h	47FFH
第三十七扇区		第三十八扇区		第三十九扇区		第四十扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
4800h	49FFh	4A00h	4BFFh	4C00h	4DFFh	4E00h	4FFFH
第四十一扇区		第四十二扇区		第四十三扇区		第四十四扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
5000h	51FFh	5200h	53FFh	5400h	55FFh	5600h	57FFH
第四十五扇区		第四十六扇区		第四十七扇区		第四十八扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
5800h	59FFh	5A00h	5BFFh	5C00h	5DFFh	5E00h	5FFFH
第四十九扇区		第五十扇区		第五十一扇区		第五十二扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
6000h	61FFh	6200h	63FFh	6400h	65FFh	6600h	67FFH
第五十三扇区		第五十四扇区		第五十五扇区		第五十六扇区	
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址
6800h	69FFh	6A00h	6BFFh	6C00h	6DFFh	6E00h	6FFFH

每个扇区512字节

建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区，不必用满，当然可全用

STC MCU Limited

STC15单片机的内部EEPROM地址表								
第五十七扇区		第五十八扇区		第五十九扇区		第六十扇区		每个扇区512字节 建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区，不必用满，当然可全用
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
7000h	71FFh	7200h	73FFh	7400h	75FFh	7600h	77FFh	
第六十一扇区		第六十二扇区		第六十三扇区		第六十四扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
7800h	79FFh	7A00h	7BFFh	7C00h	7DFFh	7E00h	7FFFh	
第六十五扇区		第六十六扇区		第六十七扇区		第六十八扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
8000h	81FFh	8200h	83FFh	8400h	85FFh	8600h	87FFh	
第六十九扇区		第七十扇区		第七十一扇区		第七十二扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
8800h	89FFh	8A00h	8BFFh	8C00h	8DFFh	8E00h	8FFFh	
第七十三扇区		第七十四扇区		第七十五扇区		第七十六扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
9000h	91FFh	9200h	93FFh	9400h	95FFh	9600h	97FFh	
第七十七扇区		第七十八扇区		第七十九扇区		第八十扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
9800h	99FFh	9A00h	9BFFh	9C00h	9DFFh	9E00h	9FFFH	
第八十一扇区		第八十二扇区		第八十三扇区		第八十四扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
A000h	A1FFh	A200h	A3FFh	A400h	A5FFh	A600h	A7FFh	
第八十五扇区		第八十六扇区		第八十七扇区		第八十八扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
A800h	A9FFh	AA00h	ABFFh	AC00h	ADFFh	AE00h	AFFFH	
第八十九扇区		第九十扇区		第九十一扇区		第九十二扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
B000h	B1FFh	B200h	B3FFh	B400h	B5FFh	B600h	B7FFh	
第九十三扇区		第九十四扇区		第九十五扇区		第九十六扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
B800h	B9FFh	BA00h	BBFFh	BC00h	BDFFh	BE00h	BFFFH	
第九十七扇区		第九十八扇区		第九十九扇区		第一百扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
C000h	C1FFh	C200h	C3FFh	C400h	C5FFh	C600h	C7FFh	
第一百零一扇区		第一百零二扇区		第一百零三扇区		第一百零四扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
C800h	C9FFh	CA00h	CBFFh	CC00h	CDFFh	CE00h	CFFFH	
第一百零五扇区		第一百零六扇区		第一百零七扇区		第一百零八扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
D000h	D1FFh	D200h	D3FFh	D400h	D5FFh	D600h	D7FFh	
第一百零九扇区		第一百一十扇区		第一百一十一扇区		第一百一十二扇区		
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
D800h	D9FFh	DA00h	DBFFh	DC00h	DDFFh	DE00h	DFFFH	

STC MCU Limited

STC15单片机的内部EEPROM地址表								
第一百一十三扇区		第一百一十四扇区		第一百一十五扇区		第一百一十六扇区		每个扇区512字节
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
E000h	E1FFh	E200h	E3FFh	E400h	E5FFh	E600h	E7FFh	
第一百一十七扇区		第一百一十八扇区		第一百一十九扇区		第一百二十扇区		建议同一次修改的数据放在同一扇区，不是同一次修改的数据放在不同的扇区，不必用满， 当然可全用
起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	起始地址	结束地址	
E800h	E9FFh	EA00h	EBFFh	EC00h	EDFFh	EE00h	EFFFH	
第一百二十一扇区		第一百二十二扇区						
起始地址	结束地址	起始地址	结束地址					
F000h	F1FFh	F200h	F3FFh					

STC MCU Limited

9.3 IAP及EEPROM汇编简介

```
;用DATA还是EQU声明新增特殊功能寄存器地址要看你用的汇编器/编译器

IAP_DATA        DATA    0C2h;   或      IAP_DATA        EQU     0C2h
IAP_ADDRH       DATA    0C3h;   或      IAP_ADDRH       EQU     0C3h
IAP_ADDRL       DATA    0C4h;   或      IAP_ADDRL       EQU     0C4h
IAP_CMD         DATA    0C5h;   或      IAP_CMD         EQU     0C5h
IAP_TRIG        DATA    0C6h;   或      IAP_TRIG        EQU     0C6h
IAP_CONTR       DATA    0C7h;   或      IAP_CONTR       EQU     0C7h

;定义ISP/IAP命令及等待时间

ISP_IAP_BYTE_READ       EQU     1       ;字节读
ISP_IAP_BYTE_PROGRAM    EQU     2       ;字节编程,前提是该字节是空，0FFh
ISP_IAP_SECTOR_ERASE    EQU     3       ;扇区擦除,要某字节为空，要擦一扇区
WAIT_TIME               EQU     0       ;设置等待时间,30MHz以下0,24M以下1,
                ;20MHz以下2,12M以下3,6M以下4,3M以下5,2M以下6,1M以下7,

;字节读,也可以用MOVC指令读，但起始地址不再是0000H，而是程序存储空间结束地址的下一个地址

MOV     IAP_ADDRH,      #BYTE_ADDR_HIGH         ;送地址高字节   ┐地址需要改变时
MOV     IAP_ADDRL,      #BYTE_ADDR_LOW          ;送地址低字节   ┘才需重新送地址
MOV     IAP_CONTR,      #WAIT_TIME      ;设置等待时间           ┐此两句可以合成一句,
ORL     IAP_CONTR,      #10000000B      ;允许ISP/IAP操作        ┘并且只送一次就够了
MOV     IAP_CMD,        #ISP_IAP_BYTE_READ
                        ;送字节读命令,现有A版本每次触发前需重新送命令。
                        ;在命令不需改变时,不需重新送命令
MOV     IAP_TRIG,       #5Ah  ;先送5Ah,再送A5h到ISP/IAP触发寄存器,每次都需如此
MOV     IAP_TRIG,       #0A5h           ;送完A5h后，ISP/IAP命令立即被触发起动
;CPU等待IAP动作完成后，才会继续执行程序。
NOP                             ;数据读出到IAP_DATA寄存器后，CPU继续执行程序
MOV     A,      ISP_DATA        ;将读出的数据送往Acc
```

STC MCU Limited

```
;以下语句可不用,只是出于安全考虑而已
    MOV     IAP_CONTR,      #00000000B      ;禁止ISP/IAP操作
    MOV     IAP_CMD,        #00000000B      ;去除ISP/IAP命令
    ;MOV    IAP_TRIG,       #00000000B      ;防止ISP/IAP命令误触发
    ;MOV    IAP_ADDRH,      #0FFh           ;送地址高字节单元为FF,指向非EEPROM区
    ;MOV    IAP_ADDRL,      #0FFh           ;送地址低字节单元为FF,防止误操作

;字节编程，该字节为FFh/空时，可对其编程，否则不行,要先执行扇区擦除
    MOV     IAP_DATA,       #ONE_DATA       ;送字节编程数据到IAP_DATA,
                                            ;只有数据改变时才需重新送
    MOV     IAP_ADDRH,      #BYTE_ADDR_HIGH ;送地址高字节   ┐地址需要改变时
    MOV     IAP_ADDRL,      #BYTE_ADDR_LOW  ;送地址低字节   ┘才需重新送地址
    MOV     IAP_CONTR,      #WAIT_TIME      ;设置等待时间   ┐此两句可合成
    ORL     IAP_CONTR,      #10000000B      ;允许ISP/IAP操作┘一句,并且只送一次就够了
    MOV     IAP_CMD,        #ISP_IAP_BYTE_PROGRAM
                                            ;送字节编程命令,现有A版本每次触发前需重新送命令。
                                            ;在命令不需改变时,不需重新送命令
    MOV     IAP_TRIG,       #5Ah    ;先送5Ah,再送A5h到ISP/IAP触发寄存器,每次都需如此
    MOV     IAP_TRIG,       #0A5h   ;送完A5h后，ISP/IAP命令立即被触发起动
;CPU等待IAP动作完成后，才会继续执行程序.
    NOP                                     ;字节编程成功后，CPU继续执行程序
;以下语句可不用,只是出于安全考虑而已
    MOV     IAP_CONTR,      #00000000B      ;禁止ISP/IAP操作
    MOV     IAP_CMD,        #00000000B      ;去除ISP/IAP命令
    ;MOV    IAP_TRIG,       #00000000B      ;防止ISP/IAP命令误触发
    ;MOV    IAP_ADDRH,      #0FFh   ;送地址高字节单元为FF,;指向非EEPROM区,防止误操作
    ;MOV    IAP_ADDRL,      #0FFh   ;送地址低字节单元为FF,指向非EEPROM区,防止误操作
```

STC MCU Limited

```
;扇区擦除，没有字节擦除，只有扇区擦除，512字节/扇区,每个扇区用得越少越方便
;如果要对某个扇区进行擦除，而其中有些字节的内容需要保留，则需将其先读到单片机
;内部的RAM中保存，再将该扇区擦除，然后将须保留的数据写回该扇区，所以每个扇区
;中用的字节数越少越好，操作起来越灵活方便.
;扇区中任意一个字节的地址都是该扇区的地址,无需求出首地址.

        MOV     IAP_ADDRH,      #SECTOR_FIRST_BYTE_ADDR_HIGH ;送扇区起始地址高字节
        MOV     IAP_ADDRL,      #SECTOR_FIRST_BYTE_ADDR_LOW  ;送扇区起始地址低字节
                                                             ;地址需要改变时才需重新送地址
        MOV     IAP_CONTR,      #WAIT_TIME                   ;设置等待时间
        ORL     IAP_CONTR,      #10000000B                   ;允许ISP/IAP
        MOV     IAP_CMD,        #ISP_IAP_SECTOR_ERASE
                                                             ;送扇区擦除命令,现有A版本每次触发前需重新送命令。
                                                             ;在命令不需改变时,不需重新送命令
        MOV     IAP_TRIG,       #5Ah
                                                             ;先送5Ah,再送A5h到ISP/IAP触发寄存器,每次都需如此
        MOV     IAP_TRIG,       #0A5h           ;送完A5h后，ISP/IAP命令立即被触发起动
;CPU等待IAP动作完成后，才会继续执行程序.
        NOP                                     ;扇区擦除成功后，CPU继续执行程序
;以下语句可不用,只是出于安全考虑而已
        MOV     IAP_CONTR,      #00000000B              ;禁止ISP/IAP操作
        MOV     IAP_CMD,        #00000000B              ;去除ISP/IAP命令
        ;MOV    IAP_TRIG,       #00000000B              ;防止ISP/IAP命令误触发
        ;MOV    IAP_ADDRH,      #0FFh           ;送地址高字节单元为FF,指向非EEPROM区
        ;MOV    IAP_ADDRL,      #0FFh           ;送地址低字节单元为FF,防止误操作
```

此两句可以合成一句,并且只送一次就够了

STC MCU Limited

小常识：（STC单片机的Data Flash 当EEPROM功能使用）

3个基本命令----字节读，字节编程，扇区擦除

字节编程：将“1”写成“1”或“0”，将“0”写成“0”。**如果某字节是FFH,才可对其**进行字节编程。**如果该字节不是FFH，则须先将整个扇区擦除，因为只有“扇区擦除”才可以**将“0”变为“1”。

扇区擦除：只有“扇区擦除”才可能将“0”擦除为“1”。

大建议：

1.同一次修改的数据放在同一扇区中，不是同一次修改的数据放在**不同的扇区，就不须读出保**护。

2.如果一个扇区只用一个字节，那就是真正的EEPROM,STC单片机的Data Flash比外部EEPROM要快很多，读一个字节/编程一个字节大概是2**个时钟/55uS。**

3.如果在一个扇区中存放了大量的数据，某次只需要修改其中的一个字节或一部分字节时，则另外的不需要修改的数据须先读出放在STC单片机的RAM中，然后擦除整个扇区，再将需要保留的数据和需修改的数据**按字节逐字节写回该扇区中(只有字节写命令，无连续字节写命令)。这**时每个扇区使用的字节数是使用的越少越方便(不需读出一大堆需保留数据)。

4.**以部分字节为一组数据时，可以在该组数据起始时增加一个特殊标志字节，该特殊标志字节**用于标志该组数据是否被使用，即该组数据中的各字节是否被写入内容。其中，标志字节可以用00H表示该组数据已被使用，用FFH表示该组数据未被使用，这样用户就只需读取起始标志字节的内容就可以判断接下来所访问的一组数据是否被使用。当用户读取到某一组数据的起始标志字节为FFH，就可以往其中写入内容，并且在使用完该组数据后，须将该组数据的起始标志字节修改为00H，再向下访问。

例如，以4**个字节一组，并在每4个字节数据前都增加一个起始标志字节，现有如下数据：**00,xx,xx,xx,xx,ff xx xx xx xx, **这部分数据表示00后的4个字节数据已经被使用，而ff后的**4个字节数据未被使用，用户可以往ff后的4个字节中写入内容，写完后再将**标志ff改为00再往**下访问。

常问的问题：

1：IAP指令完成后，地址是否会自动“加1”或“减1”？

答：不会

2：送5A和A5触发后，下一次IAP命令是否还需要送5A和A5触发？

答：是，一定要。

9.4 EEPROM测试程序(C和汇编)

9.4.1 EEPROM测试程序(不用串口送出数据)(C和汇编)

1. C程序:

```
;STC15系列单片机EEPROM/IAP 功能测试程序演示
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------*/
/* --- 演示STC 15 系列单片机 EEPROM/IAP功能----------------------------------------*/
/* --- 研发顾问QQ：800003751-------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ---------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-----*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

void IapIdle();
BYTE IapReadByte(WORD addr);

#include "reg51.h"
#include "intrins.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

//-----------------------------------------

sfr     IAP_DATA    =  0xC2;        //IAP数据寄存器
sfr     IAP_ADDRH   =  0xC3;        //IAP地址寄存器高字节
sfr     IAP_ADDRL   =  0xC4;        //IAP地址寄存器低字节
sfr     IAP_CMD     =  0xC5;        //IAP命令寄存器
sfr     IAP_TRIG    =  0xC6;        //IAP命令触发寄存器
sfr     IAP_CONTR   =  0xC7;        //IAP控制寄存器

#define CMD_IDLE        0           //空闲模式
#define CMD_READ        1           //IAP字节读命令
#define CMD_PROGRAM 2               //IAP字节编程命令
#define CMD_ERASE       3           //IAP扇区擦除命令

//#define ENABLE_IAP    0x80        //if SYSCLK<30MHz
//#define ENABLE_IAP    0x81        //if SYSCLK<24MHz
#define ENABLE_IAP      0x82        //if SYSCLK<20MHz
//#define ENABLE_IAP    0x83        //if SYSCLK<12MHz
//#define ENABLE_IAP    0x84        //if SYSCLK<6MHz
```

STC MCU Limited

```
//#define  ENABLE_IAP      0x85                    //if SYSCLK<3MHz
//#define  ENABLE_IAP      0x86                    //if SYSCLK<2MHz
//#define  ENABLE_IAP      0x87                    //if SYSCLK<1MHz

//测试地址
#define   IAP_ADDRESS     0x0400

void Delay(BYTE n);
void IapIdle();
BYTE IapReadByte(WORD addr);
void IapProgramByte(WORD addr, BYTE dat);
void IapEraseSector(WORD addr);

void main()
{
        WORD i;

        P1 = 0xfe;                                  //1111,1110 系统OK
        Delay(10);                                  //延时
        IapEraseSector(IAP_ADDRESS);                //扇区擦除
        for (i=0; i<512; i++)                       //检测是否擦除成功(全FF检测)
        {
                if (IapReadByte(IAP_ADDRESS+i) != 0xff)
                goto Error;                         //如果出错, 则退出
        }
        P1 = 0xfc;                                  //1111,1100 擦除成功
        Delay(10);                                  //延时
        for (i=0; i<512; i++)                       //编程512字节
        {
                IapProgramByte(IAP_ADDRESS+i, (BYTE)i);
        }
        P1 = 0xf8;                                  //1111,1000 编程完成
        Delay(10);                                  //延时
        for (i=0; i<512; i++)                       //校验512字节
        {
                if (IapReadByte(IAP_ADDRESS+i) != (BYTE)i)
                goto Error;                         //如果校验错误, 则退出
        }
        P1 = 0xf0;                                  //1111,0000 测试完成
        while (1);
Error:
        P1 &= 0x7f;                                 //0xxx,xxxx IAP操作失败
        while (1);
}

/*---------------------------
软件延时
---------------------------*/
```

STC MCU Limited

```
void Delay(BYTE n)
{
	WORD x;

	while (n--)
	{
		x = 0;
		while (++x);
	}
}

/*---------------------------
关闭IAP
---------------------------*/
void IapIdle()
{
	IAP_CONTR	= 0;				//关闭IAP功能
	IAP_CMD		= 0;				//清除命令寄存器
	IAP_TRIG	= 0;				//清除触发寄存器
	IAP_ADDRH	= 0x80;				//将地址设置到非IAP区域
	IAP_ADDRL	= 0;
}

/*---------------------------
从ISP/IAP/EEPROM区域读取一字节
---------------------------*/
BYTE IapReadByte(WORD addr)
{
	BYTE dat;				//数据缓冲区

	IAP_CONTR = ENABLE_IAP;			//使能IAP
	IAP_CMD = CMD_READ;			//设置IAP命令
	IAP_ADDRL = addr;			//设置IAP低地址
	IAP_ADDRH = addr >> 8;			//设置IAP高地址
	IAP_TRIG = 0x5a;			//写触发命令(0x5a)
	IAP_TRIG = 0xa5;			//写触发命令(0xa5)
	_nop_();				//等待ISP/IAP/EEPROM操作完成
	dat = IAP_DATA;				//读ISP/IAP/EEPROM数据
	IapIdle();				//关闭IAP功能

	return dat;				//返回
}
```

STC MCU Limited

```
/*----------------------------
写一字节数据到ISP/IAP/EEPROM区域
----------------------------*/
void IapProgramByte(WORD addr, BYTE dat)
{
	IAP_CONTR = ENABLE_IAP;		//使能IAP
	IAP_CMD = CMD_PROGRAM;		//设置IAP命令
	IAP_ADDRL = addr;		//设置IAP低地址
	IAP_ADDRH = addr >> 8;		//设置IAP高地址
	IAP_DATA = dat;		//写ISP/IAP/EEPROM数据
	IAP_TRIG = 0x5a;		//写触发命令(0x5a)
	IAP_TRIG = 0xa5;		//写触发命令(0xa5)
	_nop_();		//等待ISP/IAP/EEPROM操作完成
	IapIdle();
}

/*----------------------------
扇区擦除
----------------------------*/
void IapEraseSector(WORD addr)
{
	IAP_CONTR = ENABLE_IAP;		//使能IAP
	IAP_CMD = CMD_ERASE;		//设置IAP命令
	IAP_ADDRL = addr;		//设置IAP低地址
	IAP_ADDRH = addr >> 8;		//设置IAP高地址
	IAP_TRIG = 0x5a;		//写触发命令(0x5a)
	IAP_TRIG = 0xa5;		//写触发命令(0xa5)
	_nop_();		//等待ISP/IAP/EEPROM操作完成
	IapIdle();
}
```

STC MCU Limited

2. 汇编程序：

```
;STC15系列单片机EEPROM/IAP 功能测试程序演示
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ------------------------------------------------------*/
/* --- 演示STC 15 系列单片机 EEPROM/IAP功能----------------------------------------------*/
/* --- 研发顾问QQ：800003751-------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-----*/
/*---------------------------------------------------------------------------------------*/


//假定测试芯片的工作频率为18.432MHz

IAP_DATA        EQU     0C2H            //IAP数据寄存器
IAP_ADDRH       EQU     0C3H            //IAP地址寄存器高字
IAP_ADDRL       EQU     0C4H            //IAP地址寄存器低字
IAP_CMD         EQU     0C5H            //IAP命令寄存器
IAP_TRIG        EQU     0C6H            //IAP命令触发寄存器
IAP_CONTR       EQU     0C7H            //IAP控制寄存器

CMD_IDLE        EQU     0               //空闲模式
CMD_READ        EQU     1               //IAP字节读命令
CMD_PROGRAM EQU     2               //IAP字节编程命令
CMD_ERASE       EQU     3               //IAP扇区擦除命令

;ENABLE_IAP    EQU     80H             //if SYSCLK<30MHz
;ENABLE_IAP    EQU     81H             //if SYSCLK<24MHz
ENABLE_IAP      EQU     82H             //if SYSCLK<20MHz
;ENABLE_IAP    EQU     83H             //if SYSCLK<12MHz
;ENABLE_IAP    EQU     84H             //if SYSCLK<6MHz
;ENABLE_IAP    EQU     85H             //if SYSCLK<3MHz
;ENABLE_IAP    EQU     86H             //if SYSCLK<2MHz
;ENABLE_IAP    EQU     87H             //if SYSCLK<1MHz

//测试地址
IAP_ADDRESS EQU 0400H
//-----------------------------------------
        ORG     0000H
        LJMP    MAIN
;-----------------------------------------
        ORG     0100H
MAIN:
        MOV     P1,     #0FEH           //1111,1110 系统OK
        LCALL   DELAY                   //延时
```

STC MCU Limited

```
;-------------------------------
        MOV     DPTR,   #IAP_ADDRESS            //设置ISP/IAP/EEPROM地址
        LCALL   IAP_ERASE                       //扇区擦除
;-------------------------------
        MOV     DPTR,   #IAP_ADDRESS            //设置ISP/IAP/EEPROM地址
        MOV     R0,     #0                      //检测512字节
        MOV     R1,     #2
CHECK1:                                         //检测是否擦除成功(全FF检测)
        LCALL   IAP_READ                        //读IAP数据
        CJNE    A,      #0FFH,  ERROR           //如果出错,则退出
        INC     DPTR                            //IAP地址+1
        DJNZ    R0,     CHECK1
        DJNZ    R1,     CHECK1
;-------------------------------
        MOV     P1,     #0FCH                   //1111,1100 擦除成功
        LCALL   DELAY                           //延时
;-------------------------------
        MOV     DPTR,   #IAP_ADDRESS            //设置ISP/IAP/EEPROM地址
        MOV     R0,     #0                      //编程512字节
        MOV     R1,     #2
        MOV     R2,     #0
NEXT:
        MOV     A,      R2                      //准备数据
        LCALL   IAP_PROGRAM                     //字节编程
        INC     DPTR                            //IAP地址+1
        INC     R2                              //修改测试数据
        DJNZ    R0,     NEXT
        DJNZ    R1,     NEXT
;-------------------------------
        MOV     P1,     #0F8H                   //1111,1000 编程完成
        LCALL   DELAY                           //延时
;-------------------------------
        MOV     DPTR,   #IAP_ADDRESS            //设置ISP/IAP/EEPROM地址
        MOV     R0,     #0                      //校验512字节
        MOV     R1,     #2
        MOV     R2,     #0
CHECK2:
        LCALL   IAP_READ                        //读IAP数据
        CJNE    A,      2,      ERROR           //如果出错,则退出
        INC     DPTR                            //IAP地址+1
        INC     R2
        DJNZ    R0,     CHECK2
        DJNZ    R1,     CHECK2
;-------------------------------
        MOV     P1,     #0F0H                   //1111,0000 测试完成
        SJMP    $
;-------------------------------
```

STC MCU Limited

```
ERROR:
        MOV     P0,     R0
        MOV     P2,     R1
        MOV     P3,     R2
        CLR     P1.7                    //0xxx,xxxx IAP 测试失败
        SJMP    $

/*----------------------------
软件延时
----------------------------*/
DELAY:
        CLR     A
        MOV     R0,     A
        MOV     R1,     A
        MOV     R2,     #20H
DELAY1:
        DJNZ    R0,     DELAY1
        DJNZ    R1,     DELAY1
        DJNZ    R2,     DELAY1
        RET

/*----------------------------
关闭IAP
----------------------------*/
IAP_IDLE:
        MOV     IAP_CONTR,      #0              //关闭IAP功能
        MOV     IAP_CMD,        #0              //清除命令寄存器
        MOV     IAP_TRIG,       #0              //清除触发寄存器
        MOV     IAP_ADDRH,      #80H            //将地址设置到非IAP区域
        MOV     IAP_ADDRL,      #0
        RET

/*----------------------------
从ISP/IAP/EEPROM区域读取一字节
----------------------------*/
IAP_READ:
        MOV     IAP_CONTR,      #ENABLE_IAP     //使能IAP
        MOV     IAP_CMD,        #CMD_READ       //设置IAP命令
        MOV     IAP_ADDRL,DPL                   //设置IAP低地址
        MOV     IAP_ADDRH,      DPH             //设置IAP高地址
        MOV     IAP_TRIG,       #5AH            //写触发命令(0x5a)
        MOV     IAP_TRIG,       #0A5H           //写触发命令(0xa5)
        NOP                                     //等待ISP/IAP/EEPROM操作完成
        MOV     A,      IAP_DATA                //度IAP数据
```

STC MCU Limited

```
        LCALL   IAP_IDLE                        //关闭IAP功能
        RET

/*---------------------------
写一字节数据到ISP/IAP/EEPROM区域
---------------------------*/
IAP_PROGRAM:
        MOV     IAP_CONTR,      #ENABLE_IAP     //使能IAP
        MOV     IAP_CMD,        #CMD_PROGRAM    //设置IAP命令
        MOV     IAP_ADDRL,      DPL             //设置IAP低地址
        MOV     IAP_ADDRH,      DPH             //设置IAP高地址
        MOV     IAP_DATA,       A               //写IAP数据
        MOV     IAP_TRIG,       #5AH            //写触发命令(0x5a)
        MOV     IAP_TRIG,       #0A5H           //写触发命令(0xa5)
        NOP                                     //等待ISP/IAP/EEPROM操作完成
        LCALL   IAP_IDLE                        //关闭IAP功能
        RET

/*---------------------------
扇区擦除
---------------------------*/
IAP_ERASE:
        MOV     IAP_CONTR,      #ENABLE_IAP     //使能IAP
        MOV     IAP_CMD,        #CMD_ERASE      //设置IAP命令
        MOV     IAP_ADDRL,      DPL             //设置IAP低地址
        MOV     IAP_ADDRH,      DPH             //设置IAP高地址
        MOV     IAP_TRIG,       #5AH            //写触发命令(0x5a)
        MOV     IAP_TRIG,       #0A5H           //写触发命令(0xa5)
        NOP                                     //等待ISP/IAP/EEPROM操作完成
        LCALL   IAP_IDLE                        //关闭IAP功能
        RET

        END
```

STC MCU Limited

9.4.2 EEPROM测试程序(使用串口送出数据)(C和汇编)

1. C程序:

```
;STC15系列单片机EEPROM/IAP 功能测试程序演示
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ------------------------------------------------------*/
/* --- 演示STC 15 系列单片机 EEPROM/IAP功能----------------------------------------------*/
/* --- 研发顾问QQ：800003751-------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-----*/
/*---------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

//-----------------------------------------

sfr     IAP_DATA    =  0xC2;        //IAP数据寄存器
sfr     IAP_ADDRH   =  0xC3;        //IAP地址寄存器高字节
sfr     IAP_ADDRL   =  0xC4;        //IAP地址寄存器低字节
sfr     IAP_CMD     =  0xC5;        //IAP命令寄存器
sfr     IAP_TRIG    =  0xC6;        //IAP命令触发寄存器
sfr     IAP_CONTR   =  0xC7;        //IAP控制寄存器

#define CMD_IDLE    0               //空闲模式
#define CMD_READ    1               //IAP字节读命令
#define CMD_PROGRAM 2               //IAP字节编程命令
#define CMD_ERASE   3               //IAP扇区擦除命令

#define URMD        0       //0:使用定时器2作为波特率发生器
                            //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                            //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器

sfr     T2H     =       0xd6;       //定时器2高8位
sfr     T2L     =       0xd7;       //定时器2低8位

sfr     AUXR    =       0x8e;   //辅助寄存器
```

STC MCU Limited

```
//#define  ENABLE_IAP      0x80            //if SYSCLK<30MHz
//#define  ENABLE_IAP      0x81            //if SYSCLK<24MHz
#define    ENABLE_IAP      0x82            //if SYSCLK<20MHz
//#define  ENABLE_IAP      0x83            //if SYSCLK<12MHz
//#define  ENABLE_IAP      0x84            //if SYSCLK<6MHz
//#define  ENABLE_IAP      0x85            //if SYSCLK<3MHz
//#define  ENABLE_IAP      0x86            //if SYSCLK<2MHz
//#define  ENABLE_IAP      0x87            //if SYSCLK<1MHz

//测试地址
#define IAP_ADDRESS 0x0400

void Delay(BYTE n);
void IapIdle();
BYTE IapReadByte(WORD addr);
void IapProgramByte(WORD addr, BYTE dat);
void IapEraseSector(WORD addr);
void InitUart();
BYTE SendData(BYTE dat);

void main()
{
    WORD i;

    P1 = 0xfe;                          //1111,1110 系统OK
    InitUart();                         //初始化串口
    Delay(10);                          //延时
    IapEraseSector(IAP_ADDRESS);        //扇区擦除
    for (i=0; i<512; i++)               //检测是否擦除成功(全FF检测)
    {
        if (SendData(IapReadByte(IAP_ADDRESS+i)) != 0xff)
        goto Error;                     //如果出错,则退出
    }
    P1 = 0xfc;                          //1111,1100 擦除成功
    Delay(10);                          //延时
    for (i=0; i<512; i++)               //编程512字节
    {
        IapProgramByte(IAP_ADDRESS+i, (BYTE)i);
    }
    P1 = 0xf8;                          //1111,1000 编程完成
    Delay(10);                          //延时
    for (i=0; i<512; i++)               //校验512字节
    {
        if (SendData(IapReadByte(IAP_ADDRESS+i)) != (BYTE)i)
        goto Error;                     //如果校验错误,则退出
    }
    P1 = 0xf0;                          //1111,0000 测试完成
    while (1);
```

STC MCU Limited

```
Error:
        P1 &= 0x7f;                                     //0xxx,xxxx IAP操作失败
        while (1);
}

/*---------------------------
软件延时
---------------------------*/
void Delay(BYTE n)
{
        WORD x;

        while (n--)
        {
                x = 0;
                while (++x);
        }
}

/*---------------------------
关闭IAP
---------------------------*/
void IapIdle()
{
        IAP_CONTR       =       0;                      //关闭IAP功能
        IAP_CMD         =       0;                      //清除命令寄存器
        IAP_TRIG        =       0;                      //清除触发寄存器
        IAP_ADDRH       =       0x80;                   //将地址设置到非IAP区域
        IAP_ADDRL       =       0;
}

/*---------------------------
从ISP/IAP/EEPROM区域读取一字节
---------------------------*/
BYTE IapReadByte(WORD addr)
{
        BYTE dat;                                       //数据缓冲区

        IAP_CONTR = ENABLE_IAP;                         //使能IAP
        IAP_CMD = CMD_READ;                             //设置IAP命令
        IAP_ADDRL = addr;                               //设置IAP低地址
        IAP_ADDRH = addr >> 8;                          //设置IAP高地址
        IAP_TRIG = 0x5a;                                //写触发命令(0x5a)
        IAP_TRIG = 0xa5;                                //写触发命令(0xa5)
        _nop_();                                        //等待ISP/IAP/EEPROM操作完成
         dat = IAP_DATA;                                //读ISP/IAP/EEPROM数据
        IapIdle();                                      //关闭IAP功能

        return dat;                                     //返回
}
```

STC MCU Limited

```
/*---------------------------
写一字节数据到ISP/IAP/EEPROM区域
---------------------------*/
void IapProgramByte(WORD addr, BYTE dat)
{
	IAP_CONTR = ENABLE_IAP;			//使能IAP
	IAP_CMD = CMD_PROGRAM;			//设置IAP命令
	IAP_ADDRL = addr;			//设置IAP低地址
	IAP_ADDRH = addr >> 8;			//设置IAP高地址
	IAP_DATA = dat;				//写ISP/IAP/EEPROM数据
	IAP_TRIG = 0x5a;			//写触发命令(0x5a)
	IAP_TRIG = 0xa5;			//写触发命令(0xa5)
	_nop_();				//等待ISP/IAP/EEPROM操作完成
	IapIdle();
}

/*---------------------------
扇区擦除
---------------------------*/
void IapEraseSector(WORD addr)
{
	IAP_CONTR = ENABLE_IAP;			//使能IAP
	IAP_CMD = CMD_ERASE;			//设置IAP命令
	IAP_ADDRL = addr;			//设置IAP低地址
	IAP_ADDRH = addr >> 8;			//设置IAP高地址
	IAP_TRIG = 0x5a;			//写触发命令(0x5a)
	IAP_TRIG = 0xa5;			//写触发命令(0xa5)
	_nop_();				//等待ISP/IAP/EEPROM操作完成
	IapIdle();
}

/*---------------------------
初始化串口
---------------------------*/
void InitUart()
{
	SCON	=	0x5a;			//设置串口为8位可变波特率
#if	URMD	==	0
	T2L	=	0xd8;			//设置波特率重装值
	T2H	=	0xff;			//115200 bps(65536-18432000/4/115200)
	AUXR	=	0x14;			//T2为1T模式, 并启动定时器2
	AUXR	|=	0x01;			//选择定时器2为串口1的波特率发生器
#elif	URMD	==	1
	AUXR	=	0x40;			//定时器1为1T模式
	TMOD	=	0x00;			//定时器1为模式0(16位自动重载)
	TL1	=	0xd8;			//设置波特率重装值
	TH1	=	0xff;			//115200 bps(65536-18432000/4/115200)
	TR1	=	1;			//定时器1开始启动
```

STC MCU Limited

```
#else
        TMOD    =       0x20;                   //设置定时器1为8位自动重装载模式
        AUXR    =       0x40;                   //定时器1为1T模式
        TH1  =  TL1  =  0xfb;                   //115200 bps(256 - 18432000/32/115200)
        TR1     =       1;
#endif
}

/*---------------------------
发送串口数据
---------------------------*/
BYTE SendData(BYTE dat)
{
        while (!TI);                            //等待前一个数据发送完成
        TI = 0;                                 //清除发送标志
        SBUF = dat;                             //发送当前数据

        return dat;
}
```

2. 汇编程序：

```
;STC15系列单片机EEPROM/IAP 功能测试程序演示
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ------------- ------------------------------------------------------*/
/* --- 演示STC 15 系列单片机 EEPROM/IAP功能-----------------------------------------------*/
/* --- 研发顾问QQ：800003751---------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ---------------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ------------------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ---------------------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-----*/
/*-------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#define   URMD   0              //0:使用定时器2作为波特率发生器
                                //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
T2H       DATA   0D6H           //定时器2高8位
T2L       DATA   0D7H           //定时器2低8位
AUXR      DATA   08EH           //辅助寄存器

IAP_DATA        EQU     0C2H            //IAP数据寄存器
IAP_ADDRH       EQU     0C3H            //IAP地址寄存器高字
```

STC MCU Limited

```
IAP_ADDRL       EQU     0C4H                    //IAP地址寄存器低字
IAP_CMD         EQU     0C5H                    //IAP命令寄存器
IAP_TRIG        EQU     0C6H                    //IAP命令触发寄存器
IAP_CONTR       EQU     0C7H                    //IAP控制寄存器

CMD_IDLE        EQU     0                       //空闲模式
CMD_READ        EQU     1                       //IAP字节读命令
CMD_PROGRAM EQU     2                       //IAP字节编程命令
CMD_ERASE       EQU     3                       //IAP扇区擦除命令

;ENABLE_IAP    EQU     80H                     //if SYSCLK<30MHz
;ENABLE_IAP    EQU     81H                     //if SYSCLK<24MHz
ENABLE_IAP      EQU     82H                     //if SYSCLK<20MHz
;ENABLE_IAP    EQU     83H                     //if SYSCLK<12MHz
;ENABLE_IAP    EQU     84H                     //if SYSCLK<6MHz
;ENABLE_IAP    EQU     85H                     //if SYSCLK<3MHz
;ENABLE_IAP    EQU     86H                     //if SYSCLK<2MHz
;ENABLE_IAP    EQU     87H                     //if SYSCLK<1MHz

//测试地址
IAP_ADDRESS EQU 0400H
//----------------------------------------
        ORG     0000H
        LJMP    MAIN
;----------------------------------------
        ORG     0100H
MAIN:
        LCALL   INIT_UART                       //初始化串口
        MOV     P1,     #0FEH                   //1111,1110 系统OK
        LCALL   DELAY                           //延时
;------------------------------
        MOV     DPTR,   #IAP_ADDRESS            //设置ISP/IAP/EEPROM地址
        LCALL   IAP_ERASE                       //扇区擦除
;------------------------------
        MOV     DPTR,   #IAP_ADDRESS            //设置ISP/IAP/EEPROM地址
        MOV     R0,     #0                      //检测512字节
        MOV     R1,     #2
CHECK1:                                         //检测是否擦除成功(全FF检测)
        LCALL   IAP_READ                        //读IAP数据
        LCALL   SEND_DATA
        CJNE    A,      #0FFH,ERROR             //如果出错,则退出
        INC     DPTR                            //IAP地址+1
        DJNZ    R0,     CHECK1
        DJNZ    R1,     CHECK1
;------------------------------
        MOV     P1,     #0FCH                   //1111,1100 擦除成功
        LCALL   DELAY                           //延时
;------------------------------
        MOV     DPTR,   #IAP_ADDRESS            //设置ISP/IAP/EEPROM地址
```

STC MCU Limited

```
        MOV     R0,     #0                      //编程512字节
        MOV     R1,     #2
        MOV     R2,     #0
NEXT:
        MOV     A,R2                            //准备数据
        LCALL   IAP_PROGRAM                     //字节编程
        INC     DPTR                            //IAP地址+1
        INC     R2                              //修改测试数据
        DJNZ    R0,     NEXT
        DJNZ    R1,     NEXT
;------------------------------
        MOV     P1,     #0F8H                   //1111,1000 编程完成
        LCALL   DELAY                           //延时
;------------------------------
        MOV     DPTR,   #IAP_ADDRESS            //设置ISP/IAP/EEPROM地址
        MOV     R0,     #0                      //校验512字节
        MOV     R1,     #2
        MOV     R2,     #0
CHECK2:
        LCALL   IAP_READ                        //读IAP数据
        LCALL   SEND_DATA
        CJNE    A,      2,      ERROR           //如果出错,则退出
        INC     DPTR                            //IAP地址+1
        INC     R2
        DJNZ    R0,     CHECK2
        DJNZ    R1,     CHECK2
;------------------------------
        MOV     P1,     #0F0H                   //1111,0000 测试完成
        SJMP    $
;------------------------------
ERROR:
        MOV     P0,     R0
        MOV     P2,     R1
        MOV     P3,     R2
        CLR     P1.7                            //0xxx,xxxx IAP 测试失败
        SJMP    $

/*----------------------------
软件延时
----------------------------*/
DELAY:
        CLR     A
        MOV     R0,     A
        MOV     R1,     A
        MOV     R2,     #20H
DELAY1:
        DJNZ    R0,     DELAY1
        DJNZ    R1,     DELAY1
        DJNZ    R2,     DELAY1
        RET
```

STC MCU Limited

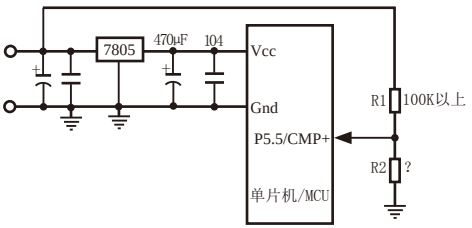
```
/*---------------------------
关闭IAP
---------------------------*/
IAP_IDLE:
        MOV     IAP_CONTR,      #0              //关闭IAP功能
        MOV     IAP_CMD,        #0              //清除命令寄存器
        MOV     IAP_TRIG,       #0              //清除触发寄存器
        MOV     IAP_ADDRH,      #80H            //将地址设置到非IAP区域
        MOV     IAP_ADDRL,      #0
        RET

/*---------------------------
从ISP/IAP/EEPROM区域读取一字节
----------------------------*/
IAP_READ:
        MOV     IAP_CONTR,      #ENABLE_IAP     //使能IAP
        MOV     IAP_CMD,        #CMD_READ       //设置IAP命令
        MOV     IAP_ADDRL,      DPL             //设置IAP低地址
        MOV     IAP_ADDRH,      DPH             //设置IAP高地址
        MOV     IAP_TRIG,       #5AH            //写触发命令(0x5a)
        MOV     IAP_TRIG,       #0A5H           //写触发命令(0xa5)
        NOP                                     //等待ISP/IAP/EEPROM操作完成
        MOV     A,          IAP_DATA            //度IAP数据
        LCALL   IAP_IDLE                        //关闭IAP功能
        RET

/*---------------------------
写一字节数据到ISP/IAP/EEPROM区域
----------------------------*/
IAP_PROGRAM:
        MOV     IAP_CONTR,      #ENABLE_IAP     //使能IAP
        MOV     IAP_CMD,        #CMD_PROGRAM    //设置IAP命令
        MOV     IAP_ADDRL,      DPL             //设置IAP低地址
        MOV     IAP_ADDRH,      DPH             //设置IAP高地址
        MOV     IAP_DATA,       A               //写IAP数据
        MOV     IAP_TRIG,       #5AH            //写触发命令(0x5a)
        MOV     IAP_TRIG,       #0A5H           //写触发命令(0xa5)
        NOP                                     //等待ISP/IAP/EEPROM操作完成
        LCALL   IAP_IDLE                        //关闭IAP功能
        RET

/*---------------------------
扇区擦除
----------------------------*/
IAP_ERASE:
        MOV     IAP_CONTR,      #ENABLE_IAP     //使能IAP
        MOV     IAP_CMD,        #CMD_ERASE      //设置IAP命令
        MOV     IAP_ADDRL,      DPL             //设置IAP低地址
        MOV     IAP_ADDRH,      DPH             //设置IAP高地址
```

STC MCU Limited

```
        MOV     IAP_TRIG,       #5AH            //写触发命令(0x5a)
        MOV     IAP_TRIG,       #0A5H           //写触发命令(0xa5)
        NOP                                     //等待ISP/IAP/EEPROM操作完成
        LCALL   IAP_IDLE                        //关闭IAP功能
        RET

;/*---------------------------
;初始化串口
;---------------------------*/
INIT_UART:
        MOV     SCON,   #5AH                    ;设置串口为8位可变波特率
#if URMD == 0
        MOV     T2L,    #0D8H                   ;设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H                    ;T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H                    ;选择定时器2为串口1的波特率发生器
#elif URMD == 1
        MOV     AUXR,   #40H                    ;定时器1为1T模式
        MOV     TMOD,   #00H                    ;定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H                   ;设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                             ;定时器1开始运行
#else
        MOV     TMOD,   #20H                    ;设置定时器1为8位自动重装载模式
        MOV     AUXR,   #40H                    ;定时器1为1T模式
        MOV     TL1,    #0FBH                   ;115200 bps(256 - 18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1
#endif
        RET

;/*---------------------------
;发送串口数据
;---------------------------*/
SEND_DATA:
        JNB     TI,     $                       ;等待前一个数据发送完成
        CLR     TI                              ;清除发送标志
        MOV     SBUF,   A                       ;发送当前数据
        RET


        END
```

STC MCU Limited

9.5 比较器作外部掉电检测的参考电路



上图中，电阻R1和R2对稳压块7805的前端电压进行分压，分压后的电压作为P5.5/CMP+的外部输入与内部BandGap参考电压(1.27V附近)进行比较。

如交流电在220V时，稳压块7805前端的直流电是11V，当交流电压降到160V时，稳压块7805前端的直流电是8.5V，电阻R1和R2就将该前端输入电压分压到CMP+端(比较器正极输入端)，此时CMP+端输入电压低于内部BandGap参考电压(1.27V附近)，可产生比较器中断，这样掉电检测时就有充足的时间将数据保存到EEPROM中。当稳压块7805前端的直流电压高于8.5V以上时，CMP+端输入电压高于内部BandGap参考电压(1.27V附近)，CPU可继续正常工作。

STC MCU Limited

第10章 STC15系列单片机的A/D转换器

下表总结了STC15系列单片机内部集成了8路10位高速A/D转换器的单片机型号：

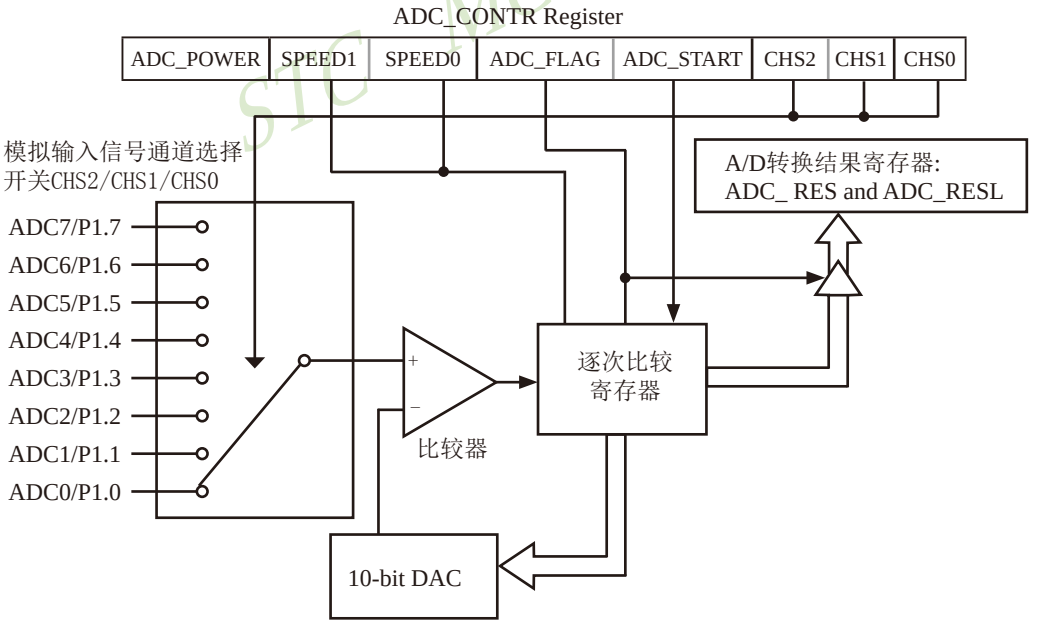
特殊外围设备 / 单片机型号	8路10位高速A/D转换器	CCP/PCA/PWM功能	1组高速同步串行口SPI
STC15W4K60S4系列	√	√	√
STC15F2K60S2系列	√	√	√
STC15W1K16S系列			√
STC15W408S系列			√
STC15W408AS系列	√	√	√
STC15W201S系列			
STC15F408AD系列	√	√	√
STC15F101W系列			

上表中 √ 表示对应的系列有相应的功能。

STC MCU Limited

10.1 A/D转换器的结构

STC15系列单片机ADC（A/D转换器）的结构如下图所示。



当CLK_DIV.5(PCON2.5)/ADRJ = 0时，A/D转换结果寄存器格式如下：

ADC_RES[7:0]	ADC_B9	ADC_B8	ADC_B7	ADC_B6	ADC_B5	ADC_B4	ADC_B3	ADC_B2	
	-	-	-	-	-	-	ADC_B1	ADC_B0	ADC_RESL[1:0]

当CLK_DIV.5(PCON2.5)/ADRJ = 1时，A/D转换结果寄存器格式如下：

ADC_RES[1:0]

-	-	-	-	-	-	ADC_B9	ADC_B8

ADC_B7	ADC_B6	ADC_B5	ADC_B4	ADC_B3	ADC_B2	ADC_B1	ADC_B0

ADC_RESL[7:0]

STC15系列单片机ADC由多路选择开关、比较器、逐次比较寄存器、10位DAC、转换结果寄存器(ADC_RES和ADC_RESL)以及ADC_CONTR构成。

STC15系列单片机的ADC是逐次比较型ADC。逐次比较型ADC由一个比较器和D/A转换器构成，通过逐次比较逻辑，从最高位(MSB)开始，顺序地对每一输入电压与内置D/A转换器输出进行比较，经过多次比较，使转换所得的数字量逐次逼近输入模拟量对应值。逐次比较型A/D转换器具有速度高，功耗低等优点。

从上图可以看出，通过模拟多路开关，将通过ADC0~7的模拟量输入送给比较器。用数/模转换器(DAC)转换的模拟量与输入的模拟量通过比较器进行比较，将比较结果保存到逐次比较寄存器，并通过逐次比较寄存器输出转换结果。A/D转换结束后，最终的转换结果保存到ADC转换结果寄存器ADC_RES和ADC_RESL，同时，置位ADC控制寄存器ADC_CONTR中的A/D转换结束标志位ADC_FLAG,以供程序查询或发出中断申请。模拟通道的选择控制由ADC控制寄存器ADC_CONTR中的CHS2~CHS0确定。ADC的转换速度由ADC控制寄存器中的SPEED1和SPEED0确定。在使用ADC之前，应先给ADC上电，也就是置位ADC控制寄存器中的ADC_POWER位。

当ADRJ=0时，如果取10位结果，则按下面公式计算：

$$\text{10-bit A/D Conversion Result:(ADC\_RES[7:0], ADC\_RESL[1:0])} = 1024 \times \frac{Vin}{Vcc}$$

当ADRJ=0时，如果取8位结果，按下面公式计算：

$$\text{8-bit A/D Conversion Result:(ADC\_RES[7:0])} = 256 \times \frac{Vin}{Vcc}$$

当ADRJ=1时，如果取10位结果，则按下面公式计算：

$$\text{10-bit A/D Conversion Result:(ADC\_RES[1:0], ADC\_RESL[7:0])} = 1024 \times \frac{Vin}{Vcc}$$

式中，Vin为模拟输入通道输入电压，Vcc为单片机实际工作电压，用单片机工作电压作为模拟参考电压。

10.2 与A/D转换相关的寄存器

与STC15系列单片机A/D转换相关的寄存器列于下表所示。

符号	描述	地址	位地址及其符号 MSB LSB								复位值
P1ASF	P1 Analog Function Configure register	9DH	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF	0000 0000B
ADC_CONTR	ADC Control Register	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0	0000 0000B
ADC_RES	ADC Result high	BDH									0000 0000B
ADC_RESL	ADC Result low	BEH									0000 0000B
CLK_DIV PCON2	时钟分频寄存器	97H	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B

1. P1口模拟功能控制寄存器P1ASF

STC15系列单片机的A/D转换口在P1口(P1.7-P1.0)，有8路10位高速A/D转换器,速度可达到300KHz(30万次/秒)。8路电压输入型A/D，可做温度检测、电池电压检测、按键扫描、频谱检测等。上电复位后P1口为弱上拉型I/O口，用户可以通过软件设置将8路中的任何一路设置为A/D转换，不需作为A/D使用的P1口可继续作为I/O口使用(建议只作为输入)。需作为A/D使用的口需先将P1ASF特殊功能寄存器中的相应位置为‘1’，将相应的口设置为模拟功能。P1ASF寄存器的格式如下：

P1ASF：P1口模拟功能控制寄存器(该寄存器是只写寄存器,读无效)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1ASF	9DH	name	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF

P1ASF[7:0]	P1.x的功能	其中P1ASF寄存器地址为：[9DH](不能够进行位寻址)
P1ASF.0 = 1	P1.0口作为模拟功能A/D使用	
P1ASF.1 = 1	P1.1口作为模拟功能A/D使用	
P1ASF.2 = 1	P1.2口作为模拟功能A/D使用	
P1ASF.3 = 1	P1.3口作为模拟功能A/D使用	
P1ASF.4 = 1	P1.4口作为模拟功能A/D使用	
P1ASF.5 = 1	P1.5口作为模拟功能A/D使用	
P1ASF.6 = 1	P1.6口作为模拟功能A/D使用	
P1ASF.7 = 1	P1.7口作为模拟功能A/D使用	

STC MCU Limited

2. ADC控制寄存器ADC_CONTR

ADC_CONTR寄存器的格式如下：

ADC_CONTR：ADC控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	name	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

对ADC_CONTR寄存器进行操作，建议直接用MOV赋值语句，不要用‘与’和‘或’语句。

ADC_POWER：ADC 电源控制位。

0：关闭ADC 电源；

1：打开A/D转换器电源.

建议进入空闲模式和掉电模式前，将ADC电源关闭，即ADC_POWER =0，可降低功耗。启动A/D转换前一定要确认A/D电源已打开，A/D转换结束后关闭A/D电源可降低功耗，也可不关闭。初次打开内部A/D转换模拟电源，需适当延时，等内部模拟电源稳定后，再启动A/D转换。

建议启动A/D转换后，在A/D转换结束之前，不改变任何I/O口的状态，有利于高精度A/D转换,如能将定时器/串行口/中断系统关闭更好。

SPEED1，SPEED0：模数转换器转换速度控制位

SPEED1	SPEED0	A/D转换所需时间
1	1	90个时钟周期转换一次，CPU工作频率21MHz时，A/D转换速度约300KHz
1	0	180个时钟周期转换一次
0	1	360个时钟周期转换一次
0	0	540个时钟周期转换一次

ADC_FLAG：模数转换器转换结束标志位,当A/D转换完成后，ADC_FLAG = 1，要由软件清0。不管是A/D 转换完成后由该位申请产生中断，还是由软件查询该标志位A/D转换是否结束,当A/D转换完成后，ADC_FLAG = 1，一定要软件清0。

ADC_START:模数转换器(ADC)转换启动控制位，设置为“1”时，开始转换,转换结束后为0。

CHS2/CHS1/CHS0：模拟输入通道选择，CHS2/CHS1/CHS0

CHS2	CHS1	CHS0	Analog Channel Select（模拟输入通道选择）
0	0	0	选择　P1.0 作为A/D输入来用
0	0	1	选择　P1.1 作为A/D输入来用
0	1	0	选择　P1.2 作为A/D输入来用
0	1	1	选择　P1.3 作为A/D输入来用
1	0	0	选择　P1.4 作为A/D输入来用
1	0	1	选择　P1.5 作为A/D输入来用
1	1	0	选择　P1.6 作为A/D输入来用
1	1	1	选择　P1.7 作为A/D输入来用

STC MCU Limited

3. ADC转换结果调整寄存器位——ADRJ

ADC转换结果调整寄存器位——ADRJ位于寄存器CLK_DIV/PCON中，用于控制ADC转换结果存放的位置。

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	0000,x000

ADRJ：ADC转换结果调整

0：ADC_RES[7:0]存放高8位ADC结果，ADC_RESL[1:0]存放低2位ADC结果

1：ADC_RES[1:0]存放高2位ADC结果，ADC_RESL[7:0]存放低8位ADC结果

4. A/D转换结果寄存器ADC_RES、ADC_RESL

特殊功能寄存器ADC_RES和ADC_RESL寄存器用于保存A/D转换结果，其格式如下：

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDh	A/D转换结果寄存器高								
ADC_RESL	BEh	A/D转换结果寄存器低								
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0

AUXR1寄存器的ADRJ位是A/D转换结果寄存器(ADC_RES,ADC_RESL)的数据格式调整控制位.

当ADRJ=0时，10位A/D转换结果的高8位存放在ADC_RES中，低2位存放在ADC_RESL的低2位中。

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDh	A/D转换结果寄存器高8位	ADC_RES9	ADC_RES8	ADC_RES7	ADC_RES6	ADC_RES5	ADC_RES4	ADC_RES3	ADC_RES2
ADC_RESL	BEh	A/D转换结果寄存器低2位	-	-	-	-	-	-	ADC_RES1	ADC_RES0
CLK_DIV (PCON2)	97H	时钟分频寄存器			ADRJ = 0					

此时，如果用户需取完整10位结果，按下面公式计算:

$$\text{10-bit A/D Conversion Result:(ADC\_RES[7:0], ADC\_RESL[1:0])} = 1024 \times \frac{Vin}{Vcc}$$

如果用户只需取8位结果，按下面公式计算:

$$\text{8-bit A/D Conversion Result:(ADC\_RES[7:0])} = 256 \times \frac{Vin}{Vcc}$$

式中，Vin为模拟输入通道输入电压，Vcc为单片机实际工作电压，用单片机工作电压作为模拟参考电压。

STC MCU Limited

当ADRJ=1时，10位A/D转换结果的高2位存放在ADC_RES的低2位中，低8位存放在ADC_RESL中。

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDh	A/D转换结果寄存器高2位	-	-	-	-	-	-	ADC_RES9	ADC_RES8
ADC_RESL	BEh	A/D转换结果寄存器低8位	ADC_RES7	ADC_RES6	ADC_RES5	ADC_RES4	ADC_RES3	ADC_RES2	ADC_RES1	ADC_RES0
CLK_DIV (PCON2)	97H	时钟分频寄存器			ADRJ = 1					

此时，如果用户需取完整10位结果，按下面公式计算:

$$\text{10-bit A/D Conversion Result:(ADC\_RES[1:0], ADC\_RESL[7:0])} = 1024 \times \frac{Vin}{Vcc}$$

式中，Vin为模拟输入通道输入电压，Vcc为单片机实际工作电压，用单片机工作电压作为模拟参考电压。

5. 中断允许寄存器IE

IE：中断允许寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的中断开放标志
EA=1，CPU开放中断，
EA=0，CPU屏蔽所有的中断申请。

EA的作用是使中断允许形成多级控制。即各中断源首先受EA控制;其次还受各中断源自己的中断允许控制位控制。

EADC：A/D转换中断允许位
EADC=1，允许A/D转换中断，
EADC=0，禁止A/D转换中断。

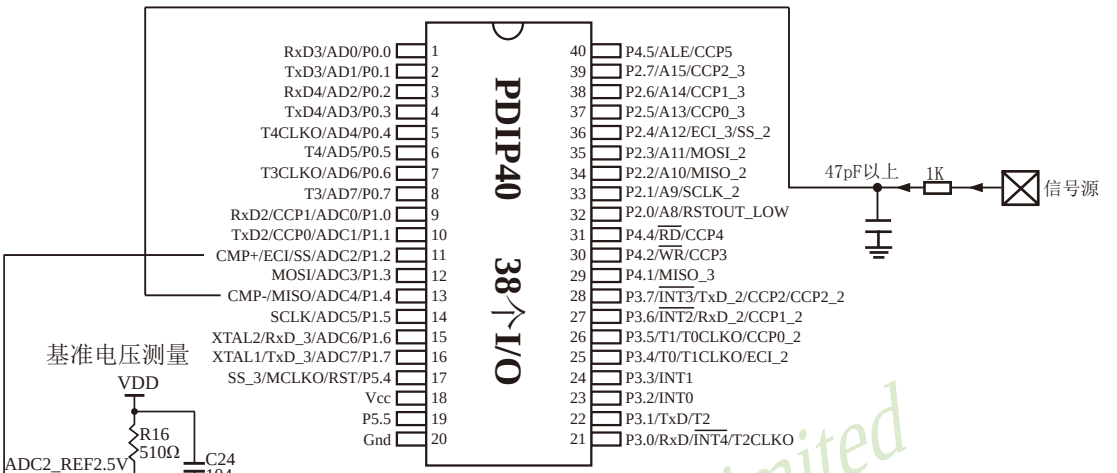
6. 中断优先级控制寄存器IP

IP：中断优先级控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PADC: A/D转换中断优先级控制位。
当PADC=0时，A/D转换中断为最低优先级中断(优先级0)
当PADC=1时，A/D转换中断为最高优先级中断(优先级1)

10.3 A/D转换典型应用线路



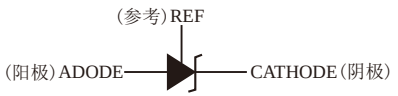
A/D转换在P1口，P1.0 ~ P1.7共8路

A/D转换通道在P1口。P1.x/ADCx是指P1.x管脚可作为A/D转换通道使用。

基准参考电压源TL431B

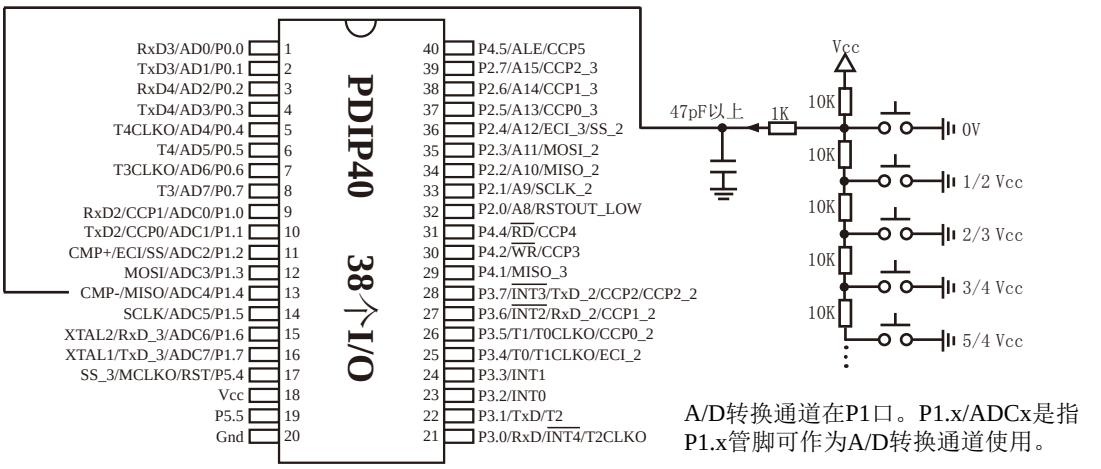
SOT23-3封装，RMB￥0.15~0.3

基准参考电压源TL431B的符号



如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

10.4 A/D作按键扫描应用线路图



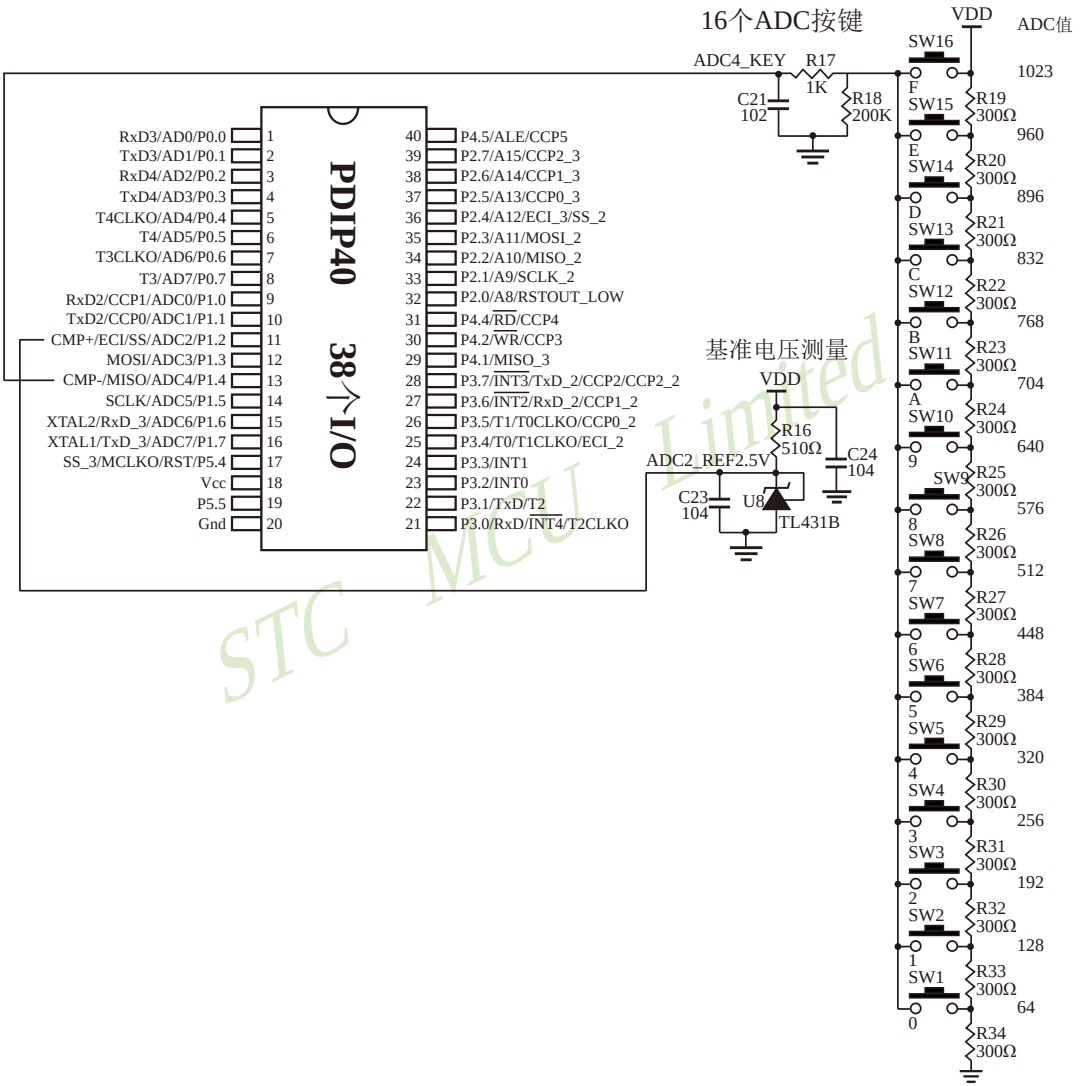
A/D转换在P1口，P1.0 ~ P1.7共8路

A/D转换通道在P1口。P1.x/ADCx是指P1.x管脚可作为A/D转换通道使用。

STC MCU Limited

读ADC键的方法：

每隔10ms左右读一次ADC值，并且保存最后3次的读数，其变化比较小时再判断键。判断键有效时，允许一定的偏差，比如±16个字的偏差。

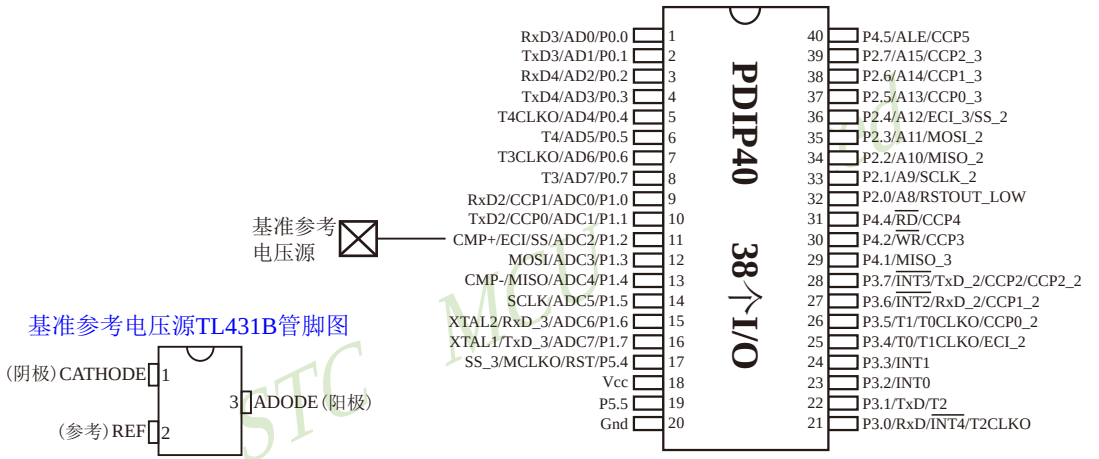


STC MCU Limited

10.5 A/D转换模块的参考电压源

STC15系列单片机的参考电压源是输入工作电压Vcc，所以一般不用外接参考电压源。如7805的输出电压是5V，但实际电压可能是4.88V到4.96V，用户需要精度比较高的话，可在出厂时将实际测出的工作电压值记录在单片机内部的EEPROM里面，以供计算。

如果有些用户的Vcc不固定，如电池供电，电池电压在5.3V-4.2V之间漂移，则Vcc不固定，就需要在8路A/D转换的一个通道外接一个稳定的参考电压源，来计算出此时的工作电压Vcc，再计算出其他几路A/D转换通道的电压。如下图所示，**可在ADC转换通道的第二通道外接一个1.25V(或1V，或．．．)的基准参考电压源，由此求出此时的工作电压Vcc，再计算出其它**几路A/D转换通道的电压(理论依据是短时间之内，Vcc不变)。

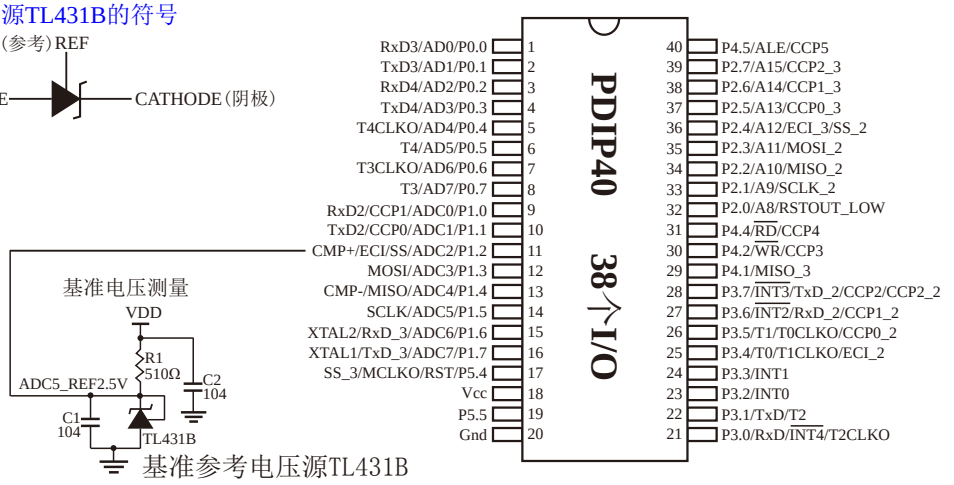


基准参考电压源TL431B管脚图

SOT23-3封装，RMB￥0.15~0.3

如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

基准参考电压源TL431B的符号



10.6 A/D转换的测试程序(C和汇编)

10.6.1 A/D转换的测试程序(ADC中断方式)

1. C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 A/D转换中断方式举例-------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可-----*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC    18432000L
#define   BAUD    9600

typedef unsigned char   BYTE;
typedef unsigned int    WORD;

#define   URMD    0             //0:使用定时器2作为波特率发生器
                                //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器

sfr     T2H     =     0xd6;     //定时器2高8位
sfr     T2L     =     0xd7;     //定时器2低8位

sfr     AUXR    =     0x8e;     //辅助寄存器

sfr     ADC_CONT    =   0xBC;   //ADC控制寄存器
sfr     ADC_RES     =   0xBD;   //ADC高8位结果
sfr     ADC_LOW2    =   0xBE;   //ADC低2位结果
sfr     P1ASF       =   0x9D;   //P1口第2功能控制寄存器

#define ADC_POWER   0x80        //ADC电源控制位
#define ADC_FLAG    0x10        //ADC完成标志
#define ADC_START   0x08        //ADC起始控制位
```

STC MCU Limited

```
#define   ADC_SPEEDLL   0x00                    //540个时钟
#define   ADC_SPEEDL    0x20                    //360个时钟
#define   ADC_SPEEDH    0x40                    //180个时钟
#define   ADC_SPEEDHH   0x60                    //90个时钟

void      InitUart();
void      SendData(BYTE dat);
void      Delay(WORD n);
void      InitADC();

BYTE      ch = 0;                                       //ADC通道号

void main()
{
          InitUart();                                   //初始化串口
          InitADC();                                    //初始化ADC
          IE = 0xa0;                                    //使能ADC中断
                                                        //开始AD转换
          while (1);
}

/*----------------------------
ADC中断服务程序
----------------------------*/
void adc_isr() interrupt 5 using 1
{
          ADC_CONTR &= !ADC_FLAG;                       //清除ADC中断标志

          SendData(ch);                                 //显示通道号
          SendData(ADC_RES);                            //读取高8位结果并发送到串口

//        SendData(ADC_LOW2);                           //显示低2位结果

          if (++ch > 7) ch = 0;                         //切换到下一个通道
          ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ADC_START | ch;
}

/*----------------------------
初始化ADC
----------------------------*/
void InitADC()
{
          P1ASF = 0xff;                                 //设置P1口为AD口
          ADC_RES = 0;                                  //清除结果寄存器
          ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ADC_START | ch;
          Delay(2);                                     //ADC上电并延时
}

/*----------------------------
```

STC MCU Limited

```
初始化串口
-----------------------------*/
void InitUart()
{
        SCON    =       0x5a;                   //设置串口为8位可变波特率
#if URMD == 0
        T2L     =       0xd8;                   //设置波特率重装值
        T2H     =       0xff;                   //115200 bps(65536-18432000/4/115200)
        AUXR    =       0x14;                   //T2为1T模式, 并启动定时器2
        AUXR    |=       0x01;                  //选择定时器2为串口1的波特率发生器
#elif   URMD    ==      1
        AUXR    =       0x40;                   //定时器1为1T模式
        TMOD    =       0x00;                   //定时器1为模式0(16位自动重载)
        TL1     =       0xd8;                   //设置波特率重装值
        TH1     =       0xff;                   //115200 bps(65536-18432000/4/115200)
        TR1     =       1;                      //定时器1开始启动
#else
        TMOD    =       0x20;                   //设置定时器1为8位自动重装载模式
        AUXR    =       0x40;                   //定时器1为1T模式
        TH1   =   TL1   =   0xfb;               //115200 bps(256 - 18432000/32/115200)
        TR1     =       1;
#endif
}

/*-----------------------------
发送串口数据
-----------------------------*/
void SendData(BYTE dat)
{
        while (!TI);                            //等待前一个数据发送完成
        TI = 0;                                 //清除发送标志
        SBUF = dat;                             //发送当前数据
}

/*-----------------------------
软件延时
-----------------------------*/
void Delay(WORD n)
{
        WORD x;

        while (n--)
        {
                x = 5000;
                while (x--);
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU  Limited. ---------------------------------------------------------------------*/
/* --- STC15F2K60S2 系列 A/D转换中断方式举例-------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/
//本示例在Keil开发环境下请选择Intel的8052芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz

#define   URMD          0       //0:使用定时器2作为波特率发生器
                                //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
T2H       DATA    0D6H          //定时器2高8位
T2L       DATA    0D7H          //定时器2低8位
AUXR      DATA    08EH          ;辅助寄存器

ADC_CONTR       EQU     0BCH        ;ADC控制寄存器
ADC_RES         EQU     0BDH        ;ADC高8位结果
ADC_LOW2        EQU     0BEH        ;ADC低2位结果
P1ASF           EQU     09DH        ;P1口第2功能控制寄存器

ADC_POWER       EQU     80H         ;ADC电源控制位
ADC_FLAG        EQU     10H         ;ADC完成标志
ADC_START       EQU     08H         ;ADC起始控制位
ADC_SPEEDLL     EQU     00H         ;540个时钟
ADC_SPEEDL      EQU     20H         ;360个时钟
ADC_SPEEDH      EQU     40H         ;180个时钟
ADC_SPEEDHH     EQU     60H         ;90个时钟

ADCCH           DATA    20H         ;ADC通道号

;------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     002BH
        LJMP    ADC_ISR
;------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH
```

```
        MOV     ADCCH,          #0
        LCALL   INIT_UART                                   ;初始化串口
        LCALL   INIT_ADC                                    ;初始化ADC
        MOV     IE,     #0A0H                               ;使能ADC中断
        SJMP    $

;/*---------------------------
;ADC中断服务程序
;---------------------------*/
ADC_ISR:
        PUSH    ACC
        PUSH    PSW

        ANL     ADC_CONTR,      #NOT ADC_FLAG               ;清除ADC中断标志
        MOV     A,              ADCCH
        LCALL   SEND_DATA                                   ;Send channel NO.
        MOV     A,              ADC_    RES                 ;Get ADC high 8-bit result
        LCALL   SEND_DATA                                   ;Send to UART

;       MOV     A,              ADC_LOW2                    ;Get ADC low 2-bit result
;       LCALL   SEND_DATA                                   ;Send to UART

        INC     ADCCH
        MOV     A,      ADCCH
        ANL     A,      #07H
        MOV     ADCCH, A
        ORL     A,      #ADC_POWER | ADC_SPEEDLL | ADC_START
        MOV     ADC_CONTR,      A                           ;AD\开始AD转换
        POP     PSW
        POP     ACC
        RETI

;/*---------------------------
;初始化ADC
;---------------------------*/
INIT_ADC:
        MOV     P1ASF,          #0FFH                       ;设置P1口为AD口
        MOV     ADC_RES,        #0                          ;清除结果寄存器
        MOV     A,              ADCCH
        ORL     A,              #ADC_POWER | ADC_SPEEDLL | ADC_START
        MOV     ADC_CONTR,      A                           ;ADC上电并延时
        MOV     A,              #2
        LCALL   DELAY
        RET

;/*---------------------------
;初始化串口
;---------------------------*/
INIT_UART:
```

STC MCU Limited

```
        MOV     SCON,   #5AH            ;设置串口为8位可变波特率
#if     URMD    ==      0
        MOV     T2L,    #0D8H           ;设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H            ;T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H            ;选择定时器2为串口1的波特率发生器
#elif URMD == 1
        MOV     AUXR,   #40H            ;定时器1为1T模式
        MOV     TMOD,   #00H            ;定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H           ;设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                     ;定时器1开始运行
#else
        MOV     TMOD,   #20H            ;设置定时器1为8位自动重装载模式
        MOV     AUXR,   #40H            ;定时器1为1T模式
        MOV     TL1,    #0FBH           ;115200 bps(256 - 18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1
#endif
        RET

;/*---------------------------
;发送串口数据
;---------------------------*/
SEND_DATA:
        JNB     TI,     $               ;等待前一个数据发送完成
        CLR     TI                      ;清除发送标志
        MOV     SBUF,   A               ;发送当前数据
        RET

;/*---------------------------
;软件延时
;---------------------------*/
DELAY:
        MOV     R2,     A
        CLR     A
        MOV     R0,     A
        MOV     R1,     A
DELAY1:
        DJNZ    R0,     DELAY1
        DJNZ    R1,     DELAY1
        DJNZ    R2,     DELAY1
        RET

        END
```

STC MCU Limited

10.6.2 A/D转换的测试程序(ADC查询方式)

1. C程序:

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ---------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 A/D转换查询方式举例-------------------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com -------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

#define    FOSC    18432000L
#define    BAUD    9600

typedef unsigned char        BYTE;
typedef unsigned int         WORD;

#define    URMD    0                 //0:使用定时器2作为波特率发生器
                                     //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                     //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器

sfr        T2H     =     0xd6;       //定时器2高8位
sfr        T2L     =     0xd7;       //定时器2低8位

sfr        AUXR    =     0x8e;       //辅助寄存器

sfr        ADC_CONTR    =   0xBC;    //ADC控制寄存器
sfr        ADC_RES      =   0xBD;    //ADC高8位结果
sfr        ADC_LOW2     =   0xBE;    //ADC低2位结果
sfr        P1ASF        =   0x9D;    /P1口第2功能控制寄存器

#define    ADC_POWER    0x80         //ADC电源控制位
#define    ADC_FLAG     0x10         //ADC完成标志
#define    ADC_START    0x08         //ADC起始控制位
#define    ADC_SPEEDLL  0x00         //540个时钟
#define    ADC_SPEEDL   0x20         //360个时钟
#define    ADC_SPEEDH   0x40         //180个时钟
#define    ADC_SPEEDHH  0x60         //90个时钟
void InitUart();
```

STC MCU Limited

```
void InitADC();
void SendData(BYTE dat);
BYTE GetADCResult(BYTE ch);
void Delay(WORD n);
void ShowResult(BYTE ch);

void main()
{
        InitUart();                     //初始化串口
        InitADC();                      //初始化ADC
        while (1)
        {
                ShowResult(0);          //显示通道0
                ShowResult(1);          //显示通道1
                ShowResult(2);          //显示通道2
                ShowResult(3);          //显示通道3
                ShowResult(4);          //显示通道4
                ShowResult(5);          //显示通道5
                ShowResult(6);          //显示通道6
                ShowResult(7);          //显示通道7
        }
}

/*----------------------------
发送ADC结果到PC
----------------------------*/
void ShowResult(BYTE ch)
{
        SendData(ch);                   //显示通道号
        SendData(GetADCResult(ch));     //显示ADC高8位结果

//      SendData(ADC_LOW2);             //显示低2位结果
}

/*----------------------------
读取ADC结果
----------------------------*/
BYTE GetADCResult(BYTE ch)
{
        ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ch | ADC_START;
        _nop_();                                        //等待4个NOP
        _nop_();
        _nop_();
        _nop_();
        while (!(ADC_CONTR & ADC_FLAG));                //等待ADC转换完成
        ADC_CONTR &= ~ADC_FLAG;                         //Close ADC

        return ADC_RES;                                 //返回ADC结果
}
```

STC MCU Limited

```
/*---------------------------
初始化串口
---------------------------*/
void InitUart()
{
        SCON    =       0x5a;                   //设置串口为8位可变波特率
#if     URMD    ==      0
        T2L     =       0xd8;                   //设置波特率重装值
        T2H     =       0xff;                   //115200 bps(65536-18432000/4/115200)
        AUXR    =       0x14;                   //T2为1T模式, 并启动定时器2
        AUXR    |=      0x01;                   //选择定时器2为串口1的波特率发生器
#elif   URMD    ==      1
        AUXR    =       0x40;                   //定时器1为1T模式
        TMOD    =       0x00;                   //定时器1为模式0(16位自动重载)
        TL1     =       0xd8;                   //设置波特率重装值
        TH1     =       0xff;                   //115200 bps(65536-18432000/4/115200)
        TR1     =       1;                      //定时器1开始启动
#else
        TMOD    =       0x20;                   //设置定时器1为8位自动重装载模式
        AUXR    =       0x40;                   //定时器1为1T模式
        TH1 =   TL1 =   0xfb;                   //115200 bps(256 - 18432000/32/115200)
        TR1     =       1;                      //定时器1开始启动
#endif
}

/*---------------------------
初始化ADC
---------------------------*/
void InitADC()
{
        P1ASF = 0xff;                           //设置P1口为AD口
        ADC_RES = 0;                            //清除结果寄存器
        ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
        Delay(2);                               //ADC上电并延时
}

/*---------------------------
发送串口数据
---------------------------*/
void SendData(BYTE dat)
{
        while (!TI);                            //等待前一个数据发送完成
        TI = 0;                                 //清除发送标志
        SBUF = dat;                             //发送当前数据
}
```

STC MCU Limited

```
/*---------------------------
软件延时
---------------------------*/
void Delay(WORD n)
{
	WORD x;

	while (n--)
	{
		x = 5000;
		while (x--);
	}
}
```

2. 汇编程序：

```
/*------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ------------------------------------------------------------------------- */
/* --- STC15F2K60S2 系列 A/D转换查询方式举例----------------------------------------------------*/
/* --- 研发顾问QQ：800003751--------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ----------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序---------------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序---------------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#define   URMD   0              //0:使用定时器2作为波特率发生器
                                //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
T2H       DATA   0D6H           //定时器2高8位
T2L       DATA   0D7H           //定时器2低8位
AUXR      DATA   08EH           ;辅助寄存器

ADC_CONTR       EQU     0BCH        ;ADC控制寄存器
ADC_RES         EQU     0BDH        ;ADC高8位结果
ADC_LOW2        EQU     0BEH        ;ADC低2位结果
P1ASF           EQU     09DH        ;P1口第2功能控制寄存器

ADC_POWER       EQU     80H         ;ADC电源控制位
ADC_FLAG        EQU     10H         ;ADC完成标志
ADC_START       EQU     08H         ;ADC起始控制位
ADC_SPEEDLL     EQU     00H         ;540个时钟
```

```
ADC_SPEEDL      EQU     20H             ;360个时钟
ADC_SPEEDH      EQU     40H             ;180个时钟
ADC_SPEEDHH     EQU     60H             ;90个时钟

;----------------------------------------
        ORG     0000H
        LJMP    MAIN
;----------------------------------------
        ORG     0100H
MAIN:
        LCALL   INIT_UART               ;初始化串口
        LCALL   INIT_ADC                ;初始化ADC
;------------------------------
NEXT:
        MOV     A,      #0
        LCALL   SHOW_RESULT             ;显示通道0的结果
        MOV     A,      #1
        LCALL   SHOW_RESULT             ;显示通道1的结果
        MOV     A,      #2
        LCALL   SHOW_RESULT             ;显示通道2的结果
        MOV     A,      #3
        LCALL   SHOW_RESULT             ;显示通道3的结果
        MOV     A,      #4
        LCALL   SHOW_RESULT             ;显示通道4的结果
        MOV     A,      #5
        LCALL SHOW_RESULT               ;显示通道5的结果
        MOV     A,      #6
        LCALL   SHOW_RESULT             ;显示通道6的结果
        MOV     A,      #7
        LCALL   SHOW_RESULT             ;显示通道7的结果

        SJMP    NEXT

;/*--------------------------
;发送ADC结果到PC
;--------------------------*/
SHOW_RESULT:
        LCALL   SEND_DATA               ;显示通道号
        LCALL   GET_ADC_RESULT          ;读取高8位结果
        LCALL   SEND_DATA               ;显示结果

;       MOV     A,      ADC_LOW2        ;读取低2位结果
;       LCALL   SEND_DATA               ;显示结果
        RET


;/*--------------------------
;读取ADC结果
;--------------------------*/
```

STC MCU Limited

```
GET_ADC_RESULT:
        ORL     A,      #ADC_POWER | ADC_SPEEDLL | ADC_START
        MOV     ADC_CONTR,      A                       ;开始AD转换
        NOP                                             ;等待4个NOP
        NOP
        NOP
        NOP
WAIT:
        MOV     A,      ADC_CONTR                       ;等待ADC转换完成
        JNB     ACC.4,  WAIT                            ;ADC_FLAG(ADC_CONTR.4)
        ANL     ADC_CONTR,      #NOT ADC_FLAG           ;清ADC标志
        MOV     A,      ADC_RES                         ;返回ADC结果
        RET

;/*----------------------------
;初始化ADC
;----------------------------*/
INIT_ADC:
        MOV     P1ASF,  #0FFH                           ;设置P1口为AD口
        MOV     ADC_RES,        #0                      ;清除结果寄存器
        MOV     ADC_CONTR,      #ADC_POWER | ADC_SPEEDLL
        MOV     A,      #2                              ;ADC上电并延时
        LCALL   DELAY
        RET

;/*----------------------------
;初始化串口
;----------------------------*/
INIT_UART:
        MOV     SCON,   #5AH                    ;设置串口为8位可变波特率
#if URMD == 0
        MOV     T2L,    #0D8H                   ;设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H                    ;T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H                    ;选择定时器2为串口1的波特率发生器
#elif URMD == 1
        MOV     AUXR,   #40H                    ;定时器1为1T模式
        MOV     TMOD,   #00H                    ;定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H                   ;设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                             ;定时器1开始运行
#else
        MOV     TMOD,   #20H                    ;设置定时器1为8位自动重装载模式
        MOV     AUXR,   #40H                    ;定时器1为1T模式
        MOV     TL1,    #0FBH                   ;115200 bps(256 - 18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1
#endif
        RET
```

STC MCU Limited

```
;/*------------------------------
;发送串口数据
;------------------------------*/
SEND_DATA:
        JNB     TI,     $               ;等待前一个数据发送完成
        CLR     TI                      ;清除发送标志
        MOV     SBUF,   A               ;发送当前数据
        RET

;/*------------------------------
;软件延时
;------------------------------*/
DELAY:
        MOV     R2,     A
        CLR     A
        MOV     R0,     A
        MOV     R1,     A
DELAY1:
        DJNZ    R0,     DELAY1
        DJNZ    R1,     DELAY1
        DJNZ    R2,     DELAY1
        RET

        END
```

STC MCU Limited

10.7 利用新增的ADC第9通道测量内部参考电压的测试程序

——所测量的内部参考电压BandGap电压用来计算工作电压Vcc

ADC的第9通道是用来测试内部BandGap参考电压的，由于内部BandGap参考电压很稳定，不会随芯片的工作电压的改变而变化，所以可以通过测量内部BandGap参考电压，然后通过ADC的值便可反推出VCC的电压，从而用户可以实现自己的低压检测功能。

ADC的第9通道的测量方法：首先将P1ASF初始化为0，即关闭所有P1口的模拟功能然后通过正常的ADC转换的方法读取第0通道的值，即可通过ADC的第9通道读取当前内部BandGap参考电压值。

用户实现自己的低压检测功能的实现方法：首先用户需要在VCC很精准的情况下(比如5.0V)，测量出内部BandGap参考电压的ADC转换值(比如为BGV5)，并将这个值保存到EEPROM中，然后在低压检测的代码中，在实际VCC变化后，测量出的内部BandGap参考电压的ADC转换值(比如为BGVx)，最后通过计算公式: 实际VCC = 5.0V * BGV5 / BGVx，即可计算出实际的VCC电压值 ，需要注意的是,第一步的BGV5的基准测量一定要精确。

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 ADC第9通道应用举例-------------------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------*/
//本示例在Keil开发环境下请选择Intel的8058芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz

//说明:
//     ADC的第9通道是用来测试内部BandGap参考电压的，由于内部BandGap参考电压很稳定，不会随芯
// 片的工作电压的改变而变化，所以可以通过测量内部BandGap参考电压，然后通过ADC的值便可反推
// 出VCC的电压，从而用户可以实现自己的低压检测功能.
//     ADC的第9通道的测量方法：首先将P1ASF初始化为0，即关闭所有P1口的模拟功能然后通过正常的
// ADC转换的方法读取第0通道的值，即可通过ADC的第9通道读取当前内部BandGap参考电压值.
//     用户实现自己的低压检测功能的实现方法：首先用户需要在VCC很精准的情况下(比如5.0V)，测量
// 出内部BandGap参考电压的ADC转换值(比如为BGV5)，并将这个值保存到EEPROM中，然后在低压检
// 测的代码中，在实际VCC变化后，测量出的内部BandGap参考电压的ADC转换值(比如为BGVx)，最后
// 通过计算公式: 实际VCC = 5.0V * BGV5 / BGVx，即可计算出实际的VCC电压值.
// 需要注意的是,第一步的BGV5的基准测量一定要精确.
```

STC MCU Limited

```
#include "reg51.h"
#include "intrins.h"

#define   FOSC    18432000L
#define   BAUD    115200

typedef   unsigned char     BYTE;
typedef   unsigned int      WORD;
#define   URMD              0       //0:使用定时器2作为波特率发生器
                                    //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                    //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
sfr       T2H               =       0xd6;           //定时器2高8位
sfr       T2L               =       0xd7;           //定时器2低8位
sfr       AUXR              =       0x8e;           //辅助寄存器

sfr       ADC_CONTR         =       0xBC;           //ADC控制寄存器
sfr       ADC_RES           =       0xBD;           //ADC高8位结果
sfr       ADC_LOW2          =       0xBE;           //ADC低2位结果
sfr       P1ASF             =       0x9D;           //P1口第2功能控制寄存器

#define   ADC_POWER   0x80                          //ADC电源控制位
#define   ADC_FLAG    0x10                          //ADC完成标志
#define   ADC_START   0x08                          //ADC起始控制位
#define   ADC_SPEEDLL 0x00                          //540个时钟
#define   ADC_SPEEDL  0x20                          //360个时钟
#define   ADC_SPEEDH  0x40                          //180个时钟
#define   ADC_SPEEDHH 0x60                          //90个时钟

void   InitUart();
void   InitADC();
void   SendData(BYTE dat);
BYTE   GetADCResult();
void   Delay(WORD n);
void   ShowResult();

void main()
{
        InitUart();                                 //初始化串口
        InitADC();                                  //初始化ADC
        while (1)
        {
                ShowResult();                       //显示ADC结果
        }
}
```

STC MCU Limited

```
/*---------------------------
发送ADC结果到PC
---------------------------*/
void ShowResult()
{
        SendData(GetADCResult());               //显示ADC高8位结果
//      SendData(ADC_LOW2);                     //显示低2位结果
}

/*---------------------------
读取ADC结果
---------------------------*/
BYTE GetADCResult()
{
        ADC_CONTR = ADC_POWER | ADC_SPEEDLL | 0 | ADC_START;
        _nop_();                                                    //等待4个NOP
        _nop_();
        _nop_();
        _nop_();
        while (!(ADC_CONTR & ADC_FLAG));                            //等待ADC转换完成
        ADC_CONTR &= ~ADC_FLAG;                                     //Close ADC

        P2 = ADC_RES;

        return ADC_RES;                                             //返回ADC结果
}

/*---------------------------
初始化串口
---------------------------*/
void InitUart()
{
        SCON    =       0x5a;                                       //设置串口为8位可变波特率
#if     URMD    ==      0
        T2L     =       0xd8;                                       //设置波特率重装值
        T2H     =       0xff;                                       //115200 bps(65536-18432000/4/115200)
        AUXR    =       0x14;                                       //T2为1T模式, 并启动定时器2
        AUXR    |=      0x01;                                       //选择定时器2为串口1的波特率发生器
#elif   URMD    ==      1
        AUXR    =       0x40;                                       //定时器1为1T模式
        TMOD    =       0x00;                                       //定时器1为模式0(16位自动重载)
        TL1     =       0xd8;                                       //设置波特率重装值
        TH1     =       0xff;                                       //115200 bps(65536-18432000/4/115200)
        TR1     =       1;                                          //定时器1开始启动
```

STC MCU Limited

```
#else
	TMOD	=	0x20;				//设置定时器1为8位自动重装载模式
	AUXR	=	0x40;				//定时器1为1T模式
	TH1	=	TL1	=	0xfb;		//115200 bps(256 - 18432000/32/115200)
	TR1 = 1;
#endif
}

/*----------------------------
初始化ADC
----------------------------*/
void InitADC()
{
	P1ASF = 0x00;					//不设置P1口为模拟口
	ADC_RES = 0;					//清除结果寄存器
	ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
	Delay(2);					//ADC上电并延时
}

/*----------------------------
发送串口数据
----------------------------*/
void SendData(BYTE dat)
{
	while (!TI);					//等待前一个数据发送完成
	TI = 0;						//清除发送标志
	SBUF = dat;					//发送当前数据
}

/*----------------------------
软件延时
----------------------------*/
void Delay(WORD n)
{
	WORD x;

	while (n--)
	{
		x = 5000;
		while (x--);
	}
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 ADC第9通道应用举例-------------------------------------------------------*/
/* --- 研发顾问QQ：800003751---------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8058芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz

//说明:
//      ADC的第9通道是用来测试内部BandGap参考电压的，由于内部BandGap参考电压很稳定，不会随芯
// 片的工作电压的改变而变化，所以可以通过测量内部BandGap参考电压，然后通过ADC的值便可反推
// 出VCC的电压，从而用户可以实现自己的低压检测功能.
//      ADC的第9通道的测量方法：首先将P1ASF初始化为0，即关闭所有P1口的模拟功能然后通过正常的
// ADC转换的方法读取第0通道的值，即可通过ADC的第9通道读取当前内部BandGap参考电压值.
//      用户实现自己的低压检测功能的实现方法：首先用户需要在VCC很精准的情况下(比如5.0V)，测量
// 出内部BandGap参考电压的ADC转换值(比如为BGV5)，并将这个值保存到EEPROM中，然后在低压检
// 测的代码中，在实际VCC变化后，测量出的内部BandGap参考电压的ADC转换值(比如为BGVx)，最后
// 通过计算公式: 实际VCC = 5.0V * BGV5 / BGVx，即可计算出实际的VCC电压值.
// 需要注意的是,第一步的BGV5的基准测量一定要精确.

#define   URMD   0          //0:使用定时器2作为波特率发生器
                            //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                            //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
T2H       DATA   0D6H       ;定时器2高8位
T2L       DATA   0D7H       ;定时器2低8位
AUXR      DATA   08EH       ;辅助寄存器

ADC_CONTR     EQU   0BCH          ;ADC控制寄存器
ADC_RES       EQU   0BDH          ;ADC高8位结果
ADC_LOW2      EQU   0BEH          ;ADC低2位结果
P1ASF         EQU   09DH          ;P1口第2功能控制寄存器

ADC_POWER     EQU   80H           ;ADC电源控制位
ADC_FLAG      EQU   10H           ;ADC完成标志
ADC_START     EQU   08H           ;ADC起始控制位
ADC_SPEEDLL   EQU   00H           ;540个时钟
```

```
ADC_SPEEDL      EQU     20H                     ;360个时钟
ADC_SPEEDH      EQU     40H                     ;180个时钟
ADC_SPEEDHH     EQU     60H                     ;90个时钟

;----------------------------------------
        ORG     0000H
        LJMP    MAIN
;----------------------------------------
        ORG     0100H
MAIN:
        LCALL   INIT_UART                       ;初始化串口
        LCALL   INIT_ADC                        ;初始化ADC
;------------------------------
NEXT:
        LCALL   SHOW_RESULT                     ;显示通道0的结果

        SJMP    NEXT

;/*--------------------------
;发送ADC结果到PC
;--------------------------*/
SHOW_RESULT:
        LCALL   GET_ADC_RESULT                          ;读取高8位结果
        LCALL   SEND_DATA                               ;显示结果

;       MOV     A,      ADC_LOW2                        ;读取低2位结果
;       LCALL   SEND_DATA                               ;显示结果
        RET



;/*--------------------------
;读取ADC结果
;--------------------------*/
GET_ADC_RESULT:
        MOV     A,      #ADC_POWER | ADC_SPEEDLL | 0 | ADC_START
        MOV     ADC_CONTR,      A                       ;开始AD转换
        NOP                                             ;等待4个NOP
        NOP
        NOP
        NOP
WAIT:
        MOV     A,      ADC_CONTR                       ;等待ADC转换完成
        JNB     ACC.4,  WAIT                            ;ADC_FLAG(ADC_CONTR.4)
        ANL     ADC_CONTR,      #NOT ADC_FLAG           ;清ADC标志
```

STC MCU Limited

```
        MOV     A,      ADC_RES                         ;返回ADC结果
        RET

;/*----------------------------
;初始化ADC
;----------------------------*/
INIT_ADC:
        MOV     P1ASF,  #00H                            ;不设置P1口为模拟口
        MOV     ADC_RES,        #0                      ;清除结果寄存器
        MOV     ADC_CONTR,      #ADC_POWER | ADC_SPEEDLL
        MOV     A,      #2                              ;ADC上电并延时
        LCALL   DELAY
        RET

;/*----------------------------
;初始化串口
;----------------------------*/
INIT_UART:
        MOV     SCON,   #5AH                    ;设置串口为8位可变波特率
#if     URMD    ==      0
        MOV     T2L,    #0D8H                   ;设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H                    ;T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H                    ;选择定时器2为串口1的波特率发生器
#elif URMD == 1
        MOV     AUXR,   #40H                    ;定时器1为1T模式
        MOV     TMOD,   #00H                    ;定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H                   ;设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                             ;定时器1开始运行
#else
        MOV     TMOD,   #20H                    ;设置定时器1为8位自动重装载模式
        MOV     AUXR,   #40H                    ;定时器1为1T模式
        MOV     TL1,    #0FBH                   ;115200 bps(256 - 18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1
#endif
        RET

;/*----------------------------
;发送串口数据
;----------------------------*/
SEND_DATA:
        JNB     TI,     $                       ;等待前一个数据发送完成
```

STC MCU Limited

```
        CLR     TI                      ;清除发送标志
        MOV     SBUF,   A               ;发送当前数据
        RET

;/*---------------------------
;软件延时
;---------------------------*/
DELAY:
        MOV     R2,     A
        CLR     A
        MOV     R0,     A
        MOV     R1,     A
DELAY1:
        DJNZ    R0,     DELAY1
        DJNZ    R1,     DELAY1
        DJNZ    R2,     DELAY1
        RET

        END
```

STC MCU Limited

10.8 利用新增的ADC第9通道测量外部电压或外部电池电压

——利用内部参考电压BandGap电压测量

ADC的第9通道是用来测试内部BandGap参考电压的，由于内部BandGap参考电压很稳定，约为1.27V，不会随芯片的工作电压的改变而变化，所以可以通过测量内部BandGap参考电压，然后通过ADC的值便可反推出外部电压或外部电池电压，从而用户可以测量外部电压或外部电池电压。

ADC的第9通道的测量方法：首先将P1ASF初始化为0，即关闭所有P1口的模拟功能然后通过正常的ADC转换的方法读取第0通道的值，即可通过ADC的第9通道读取当前内部BandGap参考电压值，约为1.27V。

测量外部电压或外部电池电压的方法：首先用户需要在外部电压或外部电池电压很精准的情况下(比如5.0V)，测量出内部BandGap参考电压的ADC转换值(比如为BGV5)，并将这个值保存到EEPROM中，然后在实际的外部电压或外部电池电压变化后，测量出的内部BandGap参考电压的ADC转换值(比如为BGVx)，最后通过计算公式: 实际外部电压或外部电池电压 = 5.0V * BGV5 / BGVx，即可计算出实际的外部电压或外部电池电压值，需要注意的是,第一步的BGV5的基准测量一定要精确。

STC MCU Limited

10.9 利用BandGap电压精确测量外部输入电压值及测试程序

ADC的第9通道是用来测试内部BandGap参考电压的，由于内部BandGap参考电压很稳定，不会随芯片的工作电压的改变而变化，所以可以通过两次测量和一次计算便可得到外部的精确电压，公式如下：

$$\frac{ADC_{bg}}{V_{bg}} = \frac{1023}{Vcc}$$

$$\frac{ADCx}{Vx} = \frac{1023}{Vcc}$$

由于两次测量的时间间隔很短,Vcc的电压在此期间的波动可忽略不计从而可推出：

$$\frac{ADC_{bg}}{V_{bg}} = \frac{ADCx}{Vx}$$ ，进一步得出 $$Vx = \frac{V_{bg} * ADCx}{ADC_{bg}}$$

其中：　ADC_{bg}为Bandgap电压的ADC测量值

V_{bg}为实际Bandgap的电压值,在单片机进行CP测试时记录的参数,单位为毫伏(mV)

ADCx为外部输入电压的ADC测量值

Vx外部输入电压的实际电压值,单位为毫伏(mV)

STC MCU Limited

具体的测试方法：首先将P1ASF初始化为0,即关闭所有P1口的模拟功能然后通过正常的ADC转换的方法读取第0通道的值,即可通过ADC的第9通道读取当前内部BandGap参考电压值ADC_{bg},然后测量有外部电压输入的ADC通道,测量出外部输入电压的ADC测量值ADCx,接下来从RAM区或者ROM区读取实际Bandgap的电压值Vbg,最后通过公式$Vx = \frac{V_{bg} * ADCx}{ADC_{bg}}$,即可计算出外部输入电压的实际电压值Vx。

利用BandGap电压精确测量外部输入电压值的测试程序如下：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ------------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 通过BandGap电压精确测量外部输入电压值举例-----*/
/* --- Mobile: (86)13922805190 ----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/
```

```
//本示例在Keil开发环境下请选择Intel的8058芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

//说明:
//    ADC的第9通道是用来测试内部BandGap参考电压的,由于内部BandGap参考电
//压很稳定,不会随芯片的工作电压的改变而变化,所以可以通过两次测量和一次计算
//便可得到外部的精确电压.公式如下:
//ADCbg / Vbg = 1023 / VCC
//ADCx / Vx   = 1023 / VCC
//由于两次测量的时间间隔很短,VCC的电压在此期间的波动可忽略不计
//从而可推出 ADCbg / Vbg = ADCx / Vx
//进一步得出 Vx = Vbg * ADCx / ADCbg
//其中:ADCbg为Bandgap电压的ADC测量值
//    Vbg为实际Bandgap的电压值,在单片机进行CP测试时记录的参数,单位为毫伏(mV)
//    ADCx为外部输入电压的ADC测量值
//    Vx外部输入电压的实际电压值,单位为毫伏(mV)
//
//具体的测试方法:首先将P1ASF初始化为0,即关闭所有P1口的模拟功能
//然后通过正常的ADC转换的方法读取第0通道的值,即可通过ADC的第9通道读取当前
//内部BandGap参考电压值ADCbg,然后测量有外部电压输入的ADC通道,测量出
//外部输入电压的ADC测量值ADCx,接下来从RAM区或者ROM区读取实际Bandgap的电压值Vbg,
//最后通过公式Vx = Vbg * ADCx / ADCbg,即可计算出外部输入电压的实际电压值Vx

//----------------------------------------

WORD idata Vbg_RAM _at_ 0xef;           //对于只有256字节RAM的MCU存放地址为0EFH
//WORD idata Vbg_RAM _at_ 0x6f;         //对于只有128字节RAM的MCU存放地址为06FH

//注意:需要在下载代码时选择"在ID号前添加重要测试参数"选项,才可在程序中获取此参数
//WORD code Vbg_ROM _at_ 0x03f7;        //1K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x07f7;        //2K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x0bf7;        //3K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x0ff7;        //4K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x13f7;        //5K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x1ff7;        //8K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x27f7;        //10K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x2ff7;        //12K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x3ff7;        //16K程序空间的MCU
```

STC MCU Limited

```
//WORD code Vbg_ROM _at_ 0x4ff7;        //20K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x5ff7;        //24K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x6ff7;        //28K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x7ff7;        //32K程序空间的MCU
//WORD code Vbg_ROM _at_ 0x9ff7;        //40K程序空间的MCU
//WORD code Vbg_ROM _at_ 0xbff7;        //48K程序空间的MCU
//WORD code Vbg_ROM _at_ 0xcff7;        //52K程序空间的MCU
//WORD code Vbg_ROM _at_ 0xdff7;        //56K程序空间的MCU
WORD code Vbg_ROM _at_ 0xeff7;          //60K程序空间的MCU

//-----------------------------------------

sfr ADC_CONTR   =   0xBC;                       //ADC控制寄存器
sfr ADC_RES     =   0xBD;                       //ADC高8位结果
sfr ADC_LOW2    =   0xBE;                       //ADC低2位结果
sfr P1ASF       =   0x9D;                       //P1口第2功能控制寄存器

#define ADC_POWER   0x80                        //ADC电源控制位
#define ADC_FLAG    0x10                        //ADC完成标志
#define ADC_START   0x08                        //ADC起始控制位
#define ADC_SPEEDLL 0x00                        //540个时钟
#define ADC_SPEEDL  0x20                        //360个时钟
#define ADC_SPEEDH  0x40                        //180个时钟
#define ADC_SPEEDHH 0x60                        //90个时钟

/*---------------------------
软件延时
---------------------------*/
void Delay(WORD n)
{
        WORD x;

        while (n--)
        {
                x = 5000;
                while (x--);
        }
}

void main()
{
        BYTE ADCbg;
        BYTE ADCx;
        WORD Vx;
```

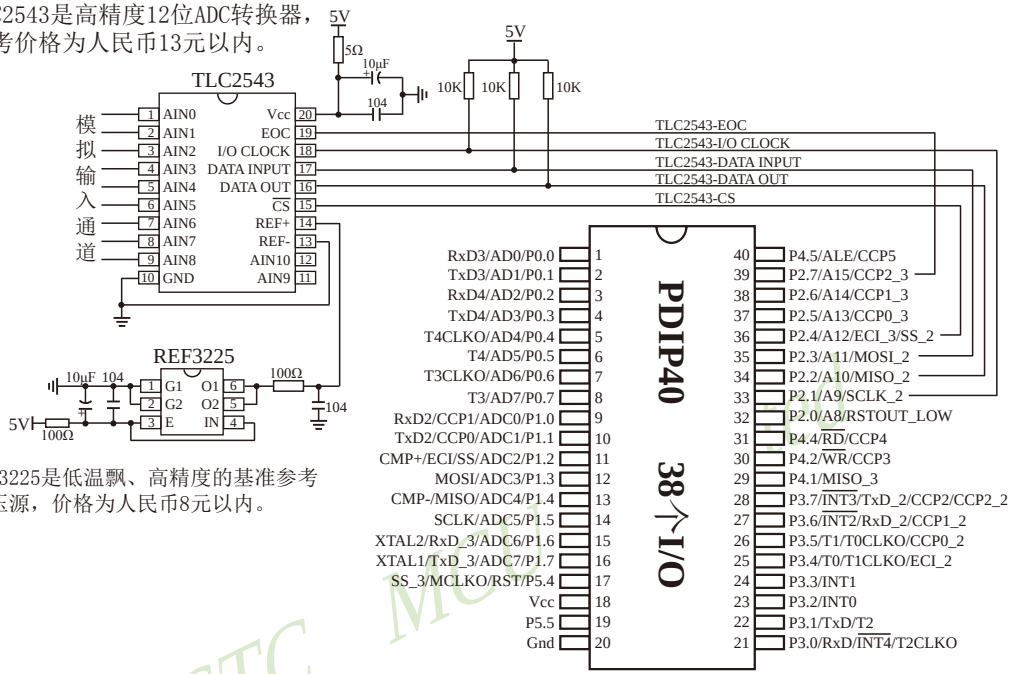
STC MCU Limited

```
//第一步:通过ADC的第9通道测试Bandgap电压的ADC测量值
        ADC_RES = 0;                        //清除结果寄存器
        P1ASF = 0x00;     //不设置P1ASF,即可从ADC的第9通道读取内部Bandgap电压的ADC测量值
        ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
        Delay(2);                           //ADC上电并延时
        ADC_CONTR = ADC_POWER | ADC_SPEEDLL | 0 | ADC_START;
        _nop_();                            //等待4个NOP
        _nop_();
        _nop_();
        _nop_();
        while (!(ADC_CONTR & ADC_FLAG));                //等待ADC转换完成
        ADC_CONTR &= ~ADC_FLAG;                         //清除ADC标志
        ADCbg = ADC_RES;

//第二步:通过ADC的第2通道测试外部输入电压的ADC测量值
        ADC_RES = 0;                                    //清除结果寄存器
        P1ASF = 0x02;                                   //设置P1.1口为模拟通道
        ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
        Delay(2);                                       //ADC上电并延时
        ADC_CONTR = ADC_POWER | ADC_SPEEDLL | 1 | ADC_START;
        _nop_();                                        //等待4个NOP
        _nop_();
        _nop_();
        _nop_();
        while (!(ADC_CONTR & ADC_FLAG));                //等待ADC转换完成
        ADC_CONTR &= ~ADC_FLAG;                         //清除ADC标志
        ADCx = ADC_RES;

//第三步:通过公式计算外部输入的实际电压值
        Vx = Vbg_RAM * ADCx / ADCbg;            //使用RAM中的Bandgap电压参数进行计算
        //Vx = Vbg_ROM * ADCx / ADCbg;          //使用ROM中的Bandgap电压参数进行计算

        while (1);
}
```

STC MCU Limited

10.10 利用SPI接口扩展12位ADC(TLC2543)的应用线路图

TLC2543是高精度12位ADC转换器，参考价格为人民币13元以内。



REF3225是低温飘、高精度的基准参考电压源，价格为人民币8元以内。

STC MCU

第11章 STC15系列CCP/PAC/PWM/DAC应用

STC15系列部分单片机集成了3路可编程计数器阵列(CCP/PCA)模块，可用于软件定时器、外部脉冲的捕捉、高速脉冲输出以及脉宽调制(PWM)输出。

下表总结了STC15系列单片机内部集成了CCP/PCA/PWM功能的单片机型号：

特殊外围设备 / 单片机型号	8路10位高速A/D转换器	CCP/PCA/PWM功能	1组高速同步串行口SPI
STC15W4K60S4系列	√	√	√
STC15F2K60S2系列	√	√	√
STC15W1K16S系列			√
STC15W408S系列			√
STC15W408AS系列	√	√	√
STC15W201S系列			
STC15F408AD系列	√	√	√
STC15F101W系列			

上表中 √ 表示对应的系列有相应的功能。

STC15W4K60S4系列、STC15F2K60S2系列和STC15F408AD系列单片机的CCP/PWM/PCA均可以在3组不同管脚之间进行切换：

[CCP0/P1.1, CCP1/P1.0, CCP2/CCP2_2/P3.7];
[CCP0_2/P3.5, CCP1_2/P3.6, CCP2/CCP2_2/P3.7];
[CCP0_3/P2.5, CCP1_3/P2.6, CCP2_3/P2.7]。

STC15W408AS系列单片机的CCP/PWM/PCA可以在2组不同管脚之间进行切换：

[CCP0/P1.1, CCP1/P1.0, CCP2/CCP2_2/P3.7];
[CCP0_2/P3.5, CCP1_2/P3.6, CCP2/CCP2_2/P3.7]。

STC15W1K16S系列、STC15W408S系列、STC15W201S系列和STC15F101W单片机没有CCP/PWM/PCA功能。

STC MCU Limited

11.1 与CCP/PWM/PCA应用有关的特殊功能寄存器

STC15系列 1T 8051单片机 CCP/PCA/PWM特殊功能寄存器表　CCP/PCA/PWM SFRs

符号	描述	地址	位地址及其符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CCON	PCA Control Register	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx,xx00
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF	0xxx,0000
CCAPM0	PCA Module 0 Mode Register	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA Module 1 Mode Register	DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA Module 2 Mode Register	DCH	-	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CL	PCA Base Timer Low	E9H									0000,0000
CH	PCA Base Timer High	F9H									0000,0000
CCAP0L	PCA Module-0 Capture Register Low	EAH									0000,0000
CCAP0H	PCA Module-0 Capture Register High	FAH									0000,0000
CCAP1L	PCA Module-1 Capture Register Low	EBH									0000,0000
CCAP1H	PCA Module-1 Capture Register High	FBH									0000,0000
CCAP2L	PCA Module-2 Capture Register Low	ECH									0000,0000
CCAP2H	PCA Module-2 Capture Register High	FCH									0000,0000
PCA_PWM0	PCA PWM Mode Auxiliary Register 0	F2H	EBS0_1	EBS0_0	-	-	-	-	EPC0H	EPC0L	00xx,xx00
PCA_PWM1	PCA PWM Mode Auxiliary Register 1	F3H	EBS1_1	EBS1_0	-	-	-	-	EPC1H	EPC1L	00xx,xx00
PCA_PWM2	PCA PWM Mode Auxiliary Register 2	F4H	EBS2_1	EBS2_0	-	-	-	-	EPC2H	EPC2L	00xx,xx00
AUXR1 P_SW1	Auxiliary Register 1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	-	DPS	0000,0000

STC MCU Limited

1. PCA工作模式寄存器CMOD

PCA工作模式寄存器的格式如下：

CMOD：PCA工作模式寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	name	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF

CIDL：空闲模式下是否停止PCA计数的控制位。

当CIDL=0时，空闲模式下PCA计数器继续工作；

当CIDL=1时，空闲模式下PCA计数器停止工作。

CPS2、CPS1、CPS0：PCA计数脉冲源选择控制位。PCA计数脉冲选择如下表所示。

CPS2	CPS1	CPS0	选择CCP/PCA/PWM时钟源输入
0	0	0	0，系统时钟，SYSclk/12
0	0	1	1，系统时钟，SYSclk/2
0	1	0	2，定时器0的溢出脉冲。由于定时器0可以工作在1T模式，所以可以达到计一个时钟就溢出，从而达到最高频率CPU工作时钟SYSclk。通过改变定时器0的溢出率，可以实现可调频率的PWM输出
0	1	1	3，ECI/P1.2(或P3.4或P2.4)脚输入的外部时钟(最大速率＝SYSclk/2)
1	0	0	4，系统时钟，SYSclk
1	0	1	5，系统时钟/4，SYSclk/4
1	1	0	6，系统时钟/6，SYSclk/6
1	1	1	7，系统时钟/8，SYSclk/8

例如，CPS2/CPS1/CPS0 = 1/0/0时，CCP/PCA/PWM的时钟源是SYSclk，不用定时器0，PWM的频率为SYSclk/256

如果要用系统时钟/3来作为PCA的时钟源,应选择T0的溢出作为CCP/PCA/PWM的时钟源，此时应让T0工作在1T模式,计数3个脉冲即产生溢出。用T0的溢出可对系统时钟进行1～65536级分频(T0工作在16位重装载模式)。

ECF：PCA计数溢出中断使能位。

当ECF = 0时，禁止寄存器CCON中CF位的中断；

当ECF = 1时，允许寄存器CCON中CF位的中断。

2. PCA控制寄存器CCON

PCA控制寄存器的格式如下：

CCON：PCA控制控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	name	CF	CR	-	-	-	CCF2	CCF1	CCF0

CF：PCA计数器阵列溢出标志位。当PCA计数器溢出时，CF由硬件置位。如果CMOD寄存器的ECF位置位， 则CF标志可用来产生中断。CF位可通过硬件或软件置位， 但只可通过软件清零。

CR：PCA计数器阵列运行控制位。该位通过软件置位， 用来起动PCA计数器阵列计数。该位通过软件清零， 用来关闭PCA计数器。

CCF2：PCA模块2中断标志。当出现匹配或捕获时该位由硬件置位。该位必须通过软件清零。

CCF1：PCA模块1中断标志。当出现匹配或捕获时该位由硬件置位。该位必须通过软件清零。

CCF0：PCA模块0中断标志。当出现匹配或捕获时该位由硬件置位。该位必须通过软件清零。

3. PCA比较/捕获寄存器CCAPM0、CCAPM1和CCAPM2

PCA模块0的比较/捕获寄存器的格式如下：

CCAPM0：PCA模块0的比较/捕获寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	DAH	name	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0

B7：保留为将来之用。

ECOM0：允许比较器功能控制位。
当ECOM0＝1时，允许比较器功能。

CAPP0：正捕获控制位。
当CAPP0＝1时，允许上升沿捕获。

CAPN0：负捕获控制位。
当CAPN0＝1时，允许下降沿捕获。

MAT0：匹配控制位。
当MAT0＝1时，PCA计数值与模块的比较/捕获寄存器的值的匹配将置位CCON寄存器的中断标志位CCF0。

TOG0：翻转控制位。
当TOG0＝1时，工作在PCA高速脉冲输出模式，PCA计数器的值与模块的比较/捕获寄存器的值的匹配将使CCP0脚翻转。
(CCP0/PCA0/PWM0/P1.1或CCP0_2/PCA0/PWM0/P3.5或CCP0_3/PCA0/PWM0/P2.5)

PWM0：脉宽调节模式。
当PWM0＝1时，允许CCP0脚用作脉宽调节输出。
(CCP0/PCA0/PWM0/P1.1或CCP0_2/PCA0/PWM0/P3.5或CCP0_3/PCA0/PWM0/P2.5)

ECCF0：使能CCF0中断。使能寄存器CCON的比较/捕获标志CCF0， 用来产生中断。

STC MCU Limited

PCA模块1的比较/捕获寄存器的格式如下：

CCAPM1：PCA模块1的比较/捕获寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM1	DBH	name	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1

B7：保留为将来之用。

ECOM1：允许比较器功能控制位。
当ECOM1＝1时，允许比较器功能。

CAPP1：正捕获控制位。
当CAPP1＝1时，允许上升沿捕获。

CAPN1：负捕获控制位。
当CAPN1＝1时，允许下降沿捕获。

MAT1：匹配控制位。
当MAT1＝1时，PCA计数值与模块的比较/捕获寄存器的值的匹配将置位CCON寄存器的中断标志位CCF1。

TOG1：翻转控制位。
当TOG1＝1时，工作在PCA高速脉冲输出模式，PCA计数器的值与模块的比较/捕获寄存器的值的匹配将使CCP1脚翻转。
(CCP1/PCA1/PWM1/P1.0或CCP1_2/PCA1/PWM1/P3.6或CCP1_3/PCA1/PWM1/P2.6)

PWM1：脉宽调节模式。
当PWM1＝1时，允许CCP1脚用作脉宽调节输出。
(CCP1/PCA1/PWM1/P1.0或CCP1_2/PCA1/PWM1/P3.6或CCP1_3/PCA1/PWM1/P2.6)

ECCF1：使能CCF1中断。使能寄存器CCON的比较/捕获标志CCF1， 用来产生中断。

PCA模块2的比较/捕获寄存器的格式如下：

CCAPM2：PCA模块2的比较/捕获寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM2	DCH	name	-	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2

B7：保留为将来之用。

ECOM2：允许比较器功能控制位。
当ECOM2＝2时，允许比较器功能。

CAPP2：正捕获控制位。
当CAPP2＝1时，允许上升沿捕获。

CAPN2：负捕获控制位。
当CAPN2＝1时，允许下降沿捕获。

STC MCU Limited

MAT2：　匹配控制位。
当MAT2＝1时，PCA计数值与模块的比较/捕获寄存器的值的匹配将置位CCON寄存器的中断标志位CCF2。

TOG2：　翻转控制位。
当TOG2＝1时，工作在PCA高速脉冲输出模式，PCA计数器的值与模块的比较/捕获寄存器的值的匹配将使CCP2脚翻转。
(CCP2/PCA2/PWM2/P3.7或CCP2/PCA2/PWM2/P2.7)

PWM2：　脉宽调节模式。
当PWM2＝1时，允许CCP2脚用作脉宽调节输出。
(CCP2/PCA2/PWM2/P3.7或CCP2/PCA2/PWM2/P2.7)

ECCF2：　使能CCF2中断。使能寄存器CCON的比较/捕获标志CCF2，　用来产生中断。

4. PCA的16位计数器 — 低8位CL和高8位CH

CL和CH地址分别为E9H和F9H，复位值均为00H，用于保存PCA的装载值。

5. PCA捕捉/比较寄存器 — CCAPnL(低位字节)和CCAPnH(高位字节)

当PCA模块用于捕获或比较时，它们用于保存各个模块的16位捕捉计数值；当PCA模块用于PWM模式时，它们用来控制输出的占空比。其中，n=0、1、2，分别对应模块0、模块1和模块2。复位值均为00H。它们对应的地址分别为：

CCAP0L — EAH、CCAP0H — FAH：模块0的捕捉/比较寄存器。

CCAP1L — EBH、CCAP1H — FBH：模块1的捕捉/比较寄存器。

CCAP2L — ECH、CCAP2H — FCH：模块2的捕捉/比较寄存器。

6. PCA模块PWM寄存器PCA_PWM0、PCA_PWM1和PCA_PWM2

PCA模块0的PWM寄存器的格式如下：

PCA_PWM0：PCA模块0的PWM寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	F2H	name	EBS0_1	EBS0_0	-	-	-	-	EPC0H	EPC0L

EBS0_1 , EBS0_0：当PCA模块0工作于PWM模式时的功能选择位。

0，0　：PCA模块0工作于8位PWM功能；

0，1　：PCA模块0工作于7位PWM功能；

1，0　：PCA模块0工作于6位PWM功能；

1，1　：无效，PCA模块0仍工作于8位PWM模式.

EPC0H：在PWM模式下，与CCAP0H组成9位数。

EPC0L：在PWM模式下，与CCAP0L组成9位数。

PCA模块1的PWM寄存器的格式如下：

PCA_PWM1：PCA模块1的PWM寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM1	F3H	name	EBS1_1	EBS1_0	-	-	-	-	EPC1H	EPC1L

EBS1_1 , EBS1_0：当PCA模块1工作于PWM模式时的功能选择位。

0 ， 0 ：PCA模块1工作于8位PWM功能；
0 ， 1 ：PCA模块1工作于7位PWM功能；
1 ， 0 ：PCA模块1工作于6位PWM功能；
1 ， 1 ：无效，PCA模块1仍工作于8位PWM.

EPC1H：在PWM模式下，与CCAP1H组成9位数。

EPC1L：在PWM模式下，与CCAP1L组成9位数。

PCA模块2的PWM寄存器的格式如下：

PCA_PWM2：PCA模块2的PWM寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM2	F4H	name	EBS2_1	EBS2_0	-	-	-	-	EPC2H	EPC2L

EBS2_1 , EBS2_0：当PCA模块2工作于PWM模式时的功能选择位。

0 ， 0 ：PCA模块2工作于8位PWM模式；
0 ， 1 ：PCA模块2工作于7位PWM模式；
1 ， 0 ：PCA模块2工作于6位PWM模式；
1 ， 1 ：无效，PCA模块2仍工作于8位PWM.

EPC2H：在PWM模式下，与CCAP2H组成9位数。

EPC2L：在PWM模式下，与CCAP2L组成9位数。

STC MCU Limited

PCA模块的工作模式设定表如下表所列：

PCA模块工作模式设定（CCAPMn寄存器，n = 0,1,2）

EBSn_1	EBSn_0	-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	模块功能
X	X		0	0	0	0	0	0	0	无此操作
0	0		1	0	0	0	0	1	0	8位PWM, 无中断
0	1		1	0	0	0	0	1	0	7位PWM, 无中断
1	0		1	0	0	0	0	1	0	6位PWM, 无中断
1	1		1	0	0	0	0	1	0	8位PWM, 无中断
0	0		1	1	0	0	0	1	1	8位PWM输出，由低变高可产生中断
0	1		1	1	0	0	0	1	1	7位PWM输出，由低变高可产生中断
1	0		1	1	0	0	0	1	1	6位PWM输出，由低变高可产生中断
1	1		1	1	0	0	0	1	1	8位PWM输出，由低变高可产生中断
0	0		1	0	1	0	0	1	1	8位PWM输出，由高变低可产生中断
0	1		1	0	1	0	0	1	1	7位PWM输出，由高变低可产生中断
1	0		1	0	1	0	0	1	1	6位PWM输出，由高变低可产生中断
1	1		1	0	1	0	0	1	1	8位PWM输出，由高变低可产生中断
0	0		1	1	1	0	0	1	1	8位PWM输出，由低变高或者由高变低均可产生中断
0	1		1	1	1	0	0	1	1	7位PWM输出，由低变高或者由高变低均可产生中断
1	0		1	1	1	0	0	1	1	6位PWM输出，由低变高或者由高变低均可产生中断
1	1		1	1	1	0	0	1	1	8位PWM输出，由低变高或者由高变低均可产生中断
X	X		X	1	0	0	0	0	X	16位捕获模式，由CCPn/PCAn的上升沿触发
X	X		X	0	1	0	0	0	X	16位捕获模式，由CCPn/PCAn的下降沿触发
X	X		X	1	1	0	0	0	X	16位捕获模式 由CCPn/PCAn的跳变触发
X	X		1	0	0	1	0	0	X	16位软件定时器
X	X		1	0	0	1	1	0	X	16位高速脉冲输出

STC MCU Limited

7. 将单片机的CCP/PWM/PCA功能在3组管脚之间切换的寄存器AUXR1(P_SW1)

辅助寄存器1的格式如下：

AUXR1/P_SW1：外围设备切换控制寄存器（不可位寻址）

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,0000

CCP可在3个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP可在P1/P2/P3之间来回切换
0	0	CCP在[P1.2/ECI,P1.1/CCP0,P1.0/CCP1,P3.7/CCP2]
0	1	CCP在[P3.4/ECI_2,P3.5/CCP0_2,P3.6/CCP1_2,P3.7/CCP2_2]
1	0	CCP在[P2.4/ECI_3,P2.5/CCP0_3,P2.6/CCP1_3,P2.7/CCP2_3]
1	1	无效

串口1/S1可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD,P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2,P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1] 串口1在P1口时要使用内部时钟
1	1	无效

SPI可在3个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI可在P1/P2/P4之间来回切换
0	0	SPI在[P1.2/SS,P1.3/MOSI,P1.4/MISO,P1.5/SCLK]
0	1	SPI在[P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI在[P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	无效

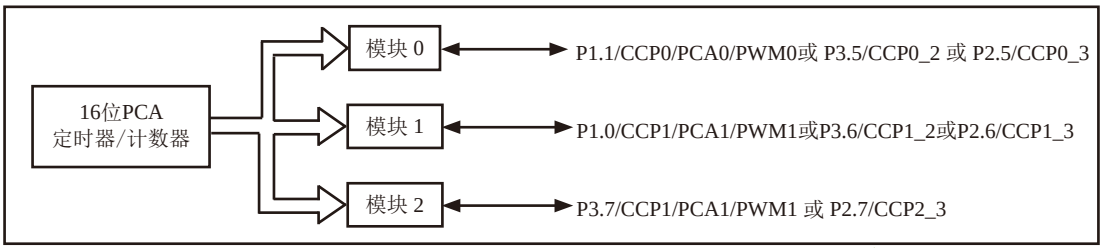
DPS：DPTR registers select bit. DPTR 寄存器选择位
　　0: DPTR0 is selected　　DPTR0被选择
　　1: DPTR1 is selected　　DPTR1被选择

STC MCU Limited

11.2 CCP/PWM/PCA模块的结构

STC15系列部分单片机有3路可编程计数器阵列CCP/PCA/PWM(通过AUXR1/P_SW1寄存器可以设置CCP/PCA/PWM从P1口切换到P2口切换到P3口）。

PCA含有一个特殊的16位定时器，有3个16位的捕获/比较模块与之相连，如下图所示。

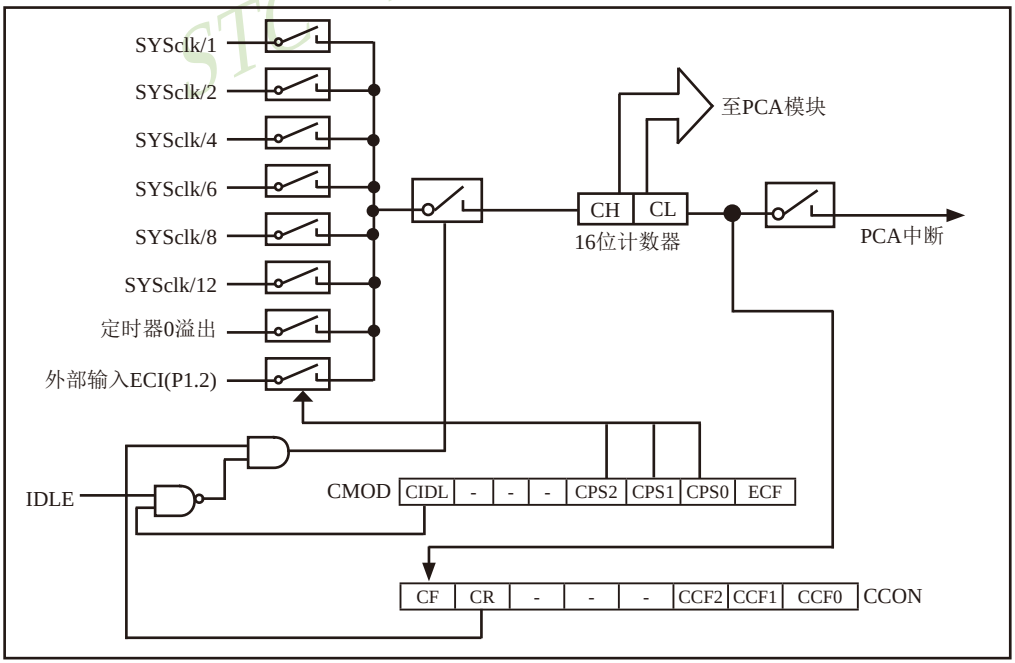


PCA模块结构

每个模块可编程工作在4种模式下：上升/下降沿捕获、软件定时器、高速脉冲输出或可调制脉冲输出。

STC15F2K60S2系列：模块0连接到P1.1/CCP0 或 P3.5/CCP0_2 或 P2.5/CCP0_3；
模块1连接到P1.0/CCP1 或 P3.6/CCP1_2 或 P2.6/CCP1_3；
模块2连接到P3.7/CCP2 或 P3.7/CCP2_2 或 P2.7/CCP2_3。

16位PCA定时器/计数器是3个模块的公共时间基准，其结构如下图所示。



PCA 定时器/计数器结构

寄存器CH和CL的内容是正在自由递增计数的16位PCA定时器的值。PCA定时器是3个模块的公共时间基准，可通过编程工作在：1/12系统时钟、1/8系统时钟、1/6系统时钟、1/4系统时钟、1/2系统时钟、系统时钟、定时器0溢出或ECI脚的输入（STC15W4K60S4系列在P1.2或P2.4或P3.4口）。定时器的计数源由CMOD特殊功能寄存器中的CPS2,CPS1和CPS0位来确定（见CMOD特殊功能寄存器说明）。

CMOD特殊功能寄存器还有2个位与PCA相关。它们分别是：CIDL，空闲模式下允许停止PCA；ECF，置位时，使能PCA中断，当PCA定时器溢出将PCA计数溢出标志CF（CCON.7）置位。

CCON特殊功能寄存器包含PCA的运行控制位（CR）和PCA定时器标志（CF）以及各个模块的标志（CCF2/CCF1/CCF0）。通过软件置位CR位（CCON.6）来运行PCA。CR位被清零时PCA关闭。当PCA计数器溢出时，CF位（CCON.7）置位,如果CMOD寄存器的ECF位置位,就产生中断。CF位只可通过软件清除。CCON寄存器的位0～2是PCA各个模块的标志（位0对应模块0，位1对应模块1，位2对应模块2），当发生匹配或比较时由硬件置位。这些标志也只能通过软件清除。所有模块共用一个中断向量。PCA的中断系统如图所示。

PCA的每个模块都对应一个特殊功能寄存器。它们分别是：模块0对应CCAPM0，模块1对应CCAPM1，模块2对应CCAPM2，特殊功能寄存器包含了相应模块的工作模式控制位。

当模块发生匹配或比较时，ECCFn位（CCAPMn.0，n＝0，1，2由工作的模块决定）使能CCON 特殊功能寄存器的CCFn标志来产生中断。

PWM（CCAPMn.1）用来使能脉宽调制模式。

当PCA计数值与模块的捕获/比较寄存器的值相匹配时，如果TOG位（CCAPMn.2）置位，模块的CCPn输出将发生翻转。

当PCA计数值与模块的捕获/比较寄存器的值相匹配时，如果匹配位MATn（CCAPMn.3）置位，CCON寄存器的CCFn位将被置位。

CAPNn（CCAPMn.4）和CAPPn（CCAPMn.5）用来设置捕获输入的有效沿。CAPNn位使能下降沿有效，CAPPn位使能上升沿有效。如果两位都置位，则两种跳变沿都被使能，捕获可在两种跳变沿产生。

通过置位CCAPMn寄存器的ECOMn位（CCAPMn.6）来使能比较器功能。

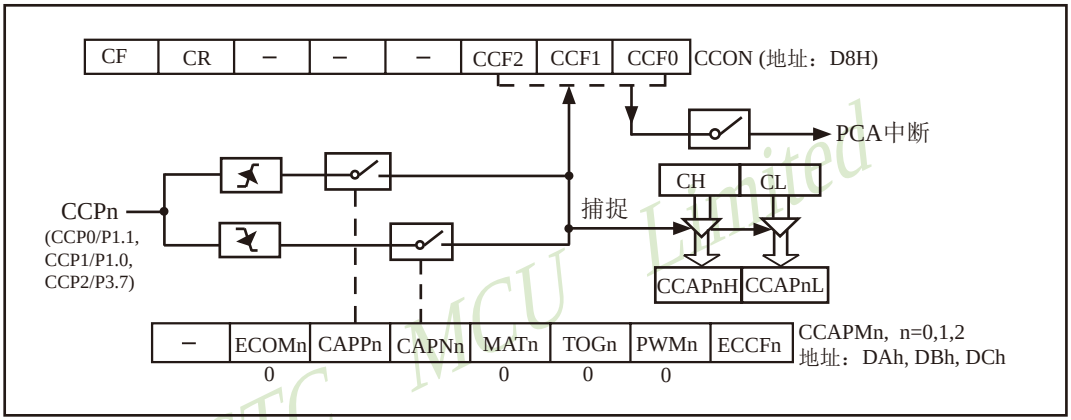
每个PCA模块还对应另外两个寄存器，CCAPnH和CCAPnL。当出现捕获或比较时，它们用来保存16位的计数值。当PCA模块用在PWM模式中时，它们用来控制输出的占空比。

STC MCU Limited

11.3 CCP/PCA模块的工作模式

11.3.1 捕获模式

PCA模块工作于捕获模式的结构图如下图所示。要使一个PCA模块工作在捕获模式，寄存器CCAPMn的两位（CAPNn和CAPPn）或其中任何一位必须置1。PCA模块工作于捕获模式时，对模块的外部CCPn输入（CCP0/P1.1,CCP1/P1.0, CCP2/P3.7）的跳变进行采样。当采样到有效跳变时，PCA硬件就将PCA计数器阵列寄存器（CH和CL）的值装载到模块的捕获寄存器中（CCAPnL和CCAPnH）。

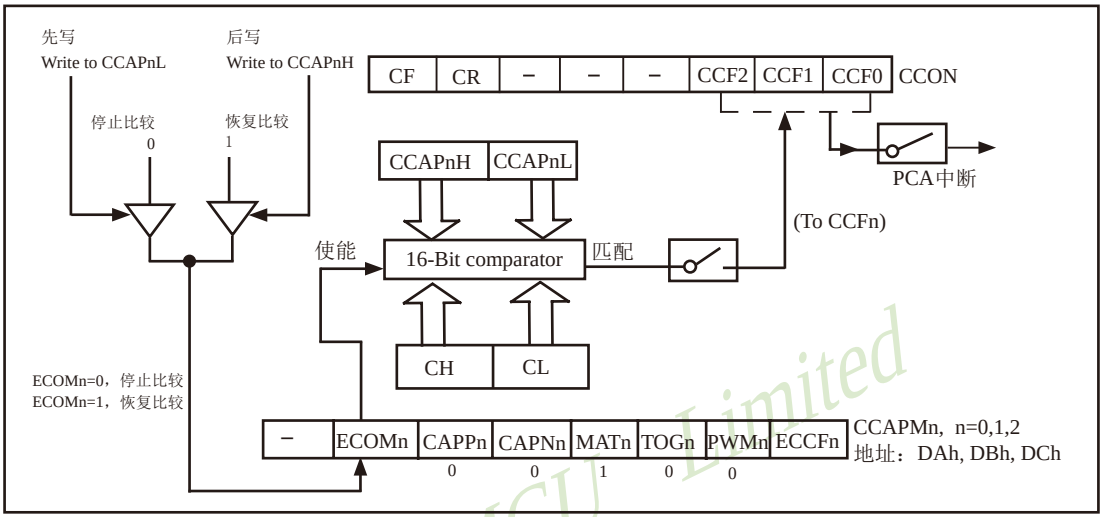


PCA Capture Mode (PCA捕获模式图)

如果CCON特殊功能寄存器中的位CCFn和CCAPMn特殊功能寄存器中的位ECCFn位被置位，将产生中断。可在中断服务程序中判断哪一个模块产生了中断，并注意中断标志位的软件清零问题。

11.3.2 16位软件定时器模式

16位软件定时器模式结构图如下图所示。



PCA Software Timer Mode / PCA模块的16位软件定时器模式/PCA比较模式

通过置位CCAPMn寄存器的ECOM和MAT位，可使PCA模块用作软件定时器（上图）。PCA定时器的值与模块捕获寄存器的值相比较，当两者相等时，如果位CCFn(在CCON特殊功能寄存器中)和位ECCFn（在CCAPMn特殊功能寄存器中）都置位，将产生中断。

[CH,CL]每隔一定的时间自动加1，时间间隔取决于选择的时钟源。例如，当选择的时钟源为SYSclk/12，每12个时钟周期[CH,CL]加1。当[CH,CL]增加到等于[CCAPnH, CCAPnL]时，CCFn=1，产生中断请求。如果每次PCA模块中断后，在中断服务程序中断给[CCAPnH, CCAPnL]增加一个相同的数值，那么下次中断来临的间隔时间T也是相同的，从而实现了定时功能。定时时间的长短取决于时钟源的选择以及PCA计数器计数值的设置。下面举例说明PCA计数器计数值的计算方法。

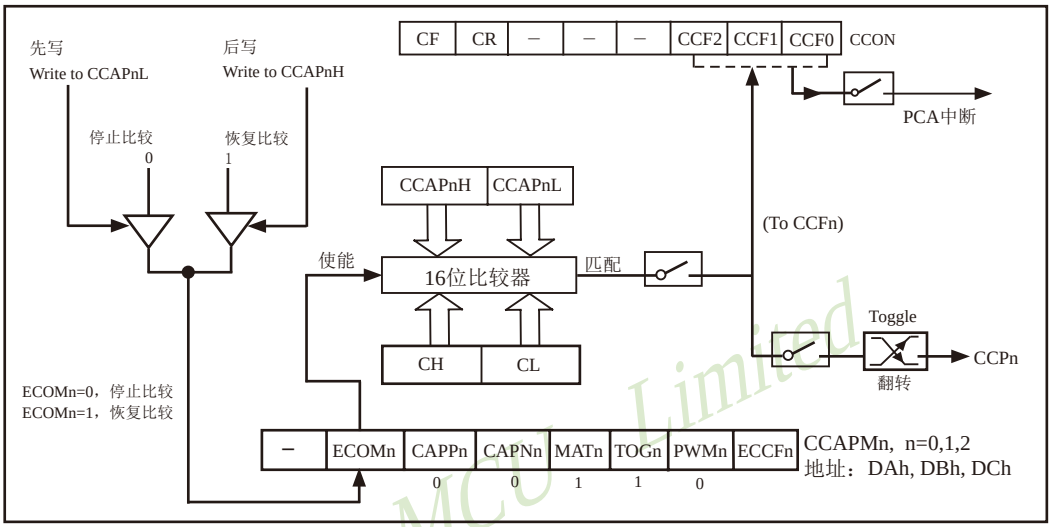
假设，系统时钟频率SYSclk = 18.432MHz, 选择的时钟源为SYSclk/12, 定时时间T为5ms, 则PCA计数器计数值为：

PCA计数器的计数值 = T / ((1 / SYSclk)×12) = 0.005 / ((1 / 18432000)×12) = 7680 (10进制数)
= 1E00H (16进制数)

也就是说，PCA计数器计数1E00H次，定时时间才是5ms，这也就是每次给[CCAPnH, CCAPnL]增加的数值(步长)。

11.3.3 高速脉冲输出模式

该模式中(下图)，当PCA计数器的计数值与模块捕获寄存器的值相匹配时,PCA模块的CCPn输出将发生翻转。要激活高速脉冲输出模式,CCAPMn寄存器的TOGn,MATn和ECOMn位必须都置位。



PCA High-Speed Output Mode / PCA 高速脉冲输出模式

CCAPnL的值决定了PCA模块n的输出脉冲频率。当PCA时钟源是SYSclk/2时，输出脉冲的频率F为：

$$f = SYSclk / (4\times CCAPnL)$$

其中，SYSclk为系统时钟频率。由此，可以得到CCAPnL的值CCAPnL = SYSclk / (4×f).

如果计算出的结果不是整数，则进行四舍五入取整，即

$$CCAPnL = INT (SYSclk / (4\times f) + 0.5)$$

其中，INT( )为取整运算，直接去掉小数。例如，假设SYSclk = 20MHz，要求PCA高速脉冲输出125kHz的方波，则CCAPnL中的值应为：

$$CCAPnL = INT (20000000 / (4\times 125000) + 0.5) = INT (40 + 0.5) = 40 = 28H$$

STC MCU Limited

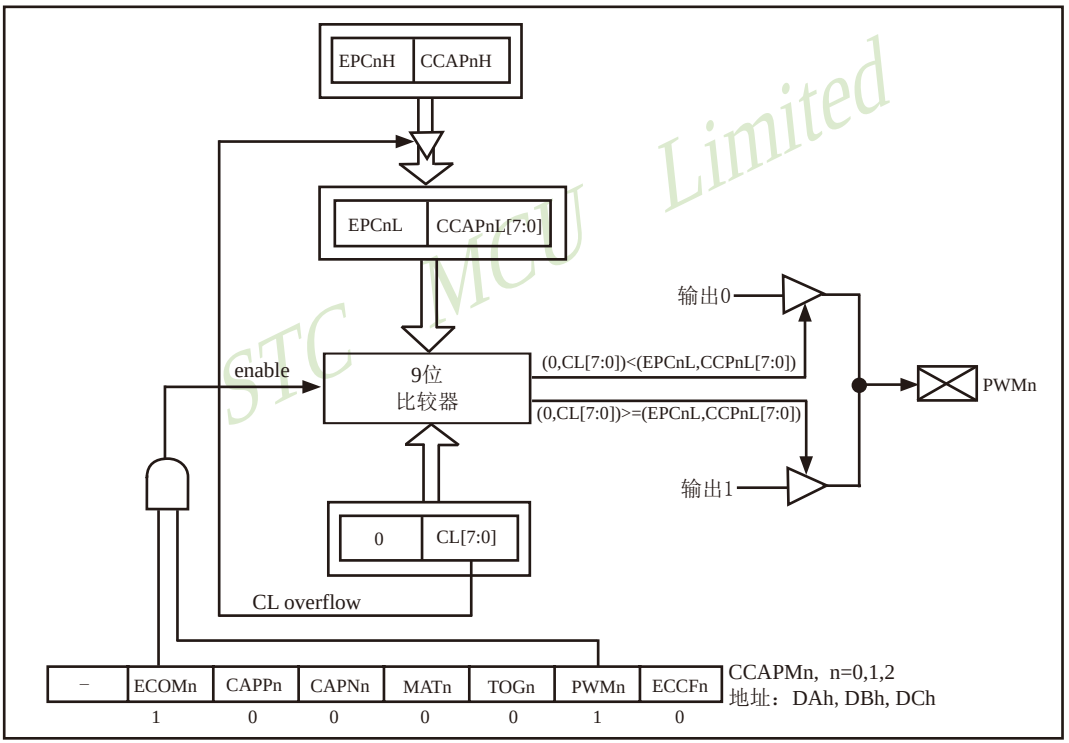
11.3.4 脉宽调节模式(PWM)

脉宽调制(PWM，Pulse Width Modulation)是一种使用程序来控制波形占空比、周期、相位波形的技术，在三相电机驱动、D/A转换等场合有广泛的应用。

STC15系列单片机的PCA模块可以通过设定各自的寄存器PCA_PWMn (n=0,1,2. 下同)中的位EBSn_1/PCA_PWMn.7及EBSn_0/PCA_PWMn.6，使其工作于8位PWM或7位PWM或6位PWM模式。

11.3.4.1 8位脉宽调节模式(PWM)

当[EBSn_1,EBSn_0]=[0,0]或[1,1]时，PCA模块n工作于8位PWM模式，此时将{0, CL[7:0]}与捕获寄存器[EPCnL, CCAPnL[7:0]]进行比较。PWM模式的结构如下图所示。



PCA PWM mode / 可调制脉冲宽度输出模式结构图(PCA模块工作于8位PWM模式)

当PCA模块工作于8位PWM模式时，由于所有模块共用仅有的PCA定时器，所有它们的输出频率相同。各个模块的输出占空比是独立变化的，与使用的捕获寄存器{EPCnL, CCAPnL[7:0]}有关。当{0, CL[7:0]}的值小于{EPCnL, CCAPnL[7:0]}时，输出为低；当{0, CL[7:0]}的值等于或大于{EPCnL, CCAPnL[7:0]}时，输出为高。当CL的值由FF变为00溢出时，{EPCnH, CCAPnH[7:0]}的内容装载到{EPCnL, CCAPnL[7:0]}中。这样就可实现无干扰地更新PWM。要使能PWM模式，模块CCAPMn寄存器的PWMn和ECOMn位必须置位。

当PWM是8位的时： PWM的频率= $\frac{\text{PCA时钟输入源频率}}{256}$

PCA时钟输入源可以从以下8种中选择一种：SYSclk， SYSclk/2， SYSclk/4，SYSclk/6，SYSclk/8，SYSclk/12, 定时器0的溢出，ECI/P1.2输入。

举例：设PCA模块工作于8位PWM模式。要求PWM输出频率为38KHz，选SYSclk为PCA/PWM时钟输入源，求出SYSclk的值。

由计算公式38000=SYSclk/256 ，得到外部时钟频率SYSclk=38000 x 256 x 1=9,728,000

如果要实现可调频率的PWM输出, 可选择定时器0的溢出率或者ECI脚的输入作为PCA/PWM的时钟输入源

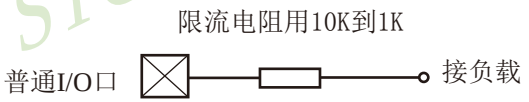
当EPCnL = 0及CCAPnL = 00H时, PWM固定输出高

当EPCnL = 1及CCAPnL = FFH时, PWM固定输出低

当某个I/O口作为PWM使用时, 该口的状态:

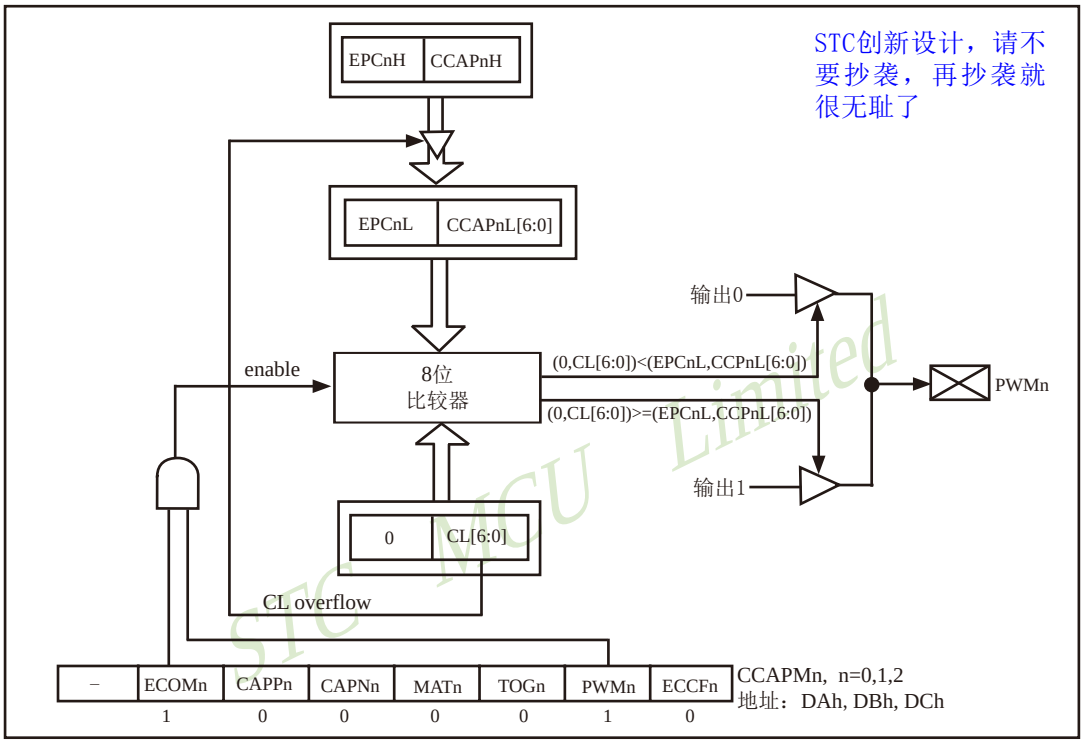
PWM之前口的状态	PWM输出时口的状态
弱上拉/准双向	强推挽输出/强上拉输出，要加输出限流电阻1K-10K
强推挽输出/强上拉输出	强推挽输出/强上拉输出，要加输出限流电阻1K-10K
仅为输入/高阻	PWM无效
开漏	开漏

STC MCU Limited



11.3.4.2 7位脉宽调节模式(PWM)(STC创新设计，请不要抄袭)

当[EBSn_1,EBSn_0]=[0,1]时，PCA模块n工作于7位PWM模式，此时将{0, CL[6:0]}与捕获寄存器[EPCnL, CCAPnL[6:0]]进行比较。PWM模式的结构如下图所示。



PCA PWM mode / 可调制脉冲宽度输出模式结构图(PCA模块工作于7位PWM模式)

当PCA模块工作于7位PWM模式时，由于所有模块共用仅有的PCA定时器，所有它们的输出频率相同。各个模块的输出占空比是独立变化的，与使用的捕获寄存器{EPCnL, CCAPnL[6:0]}有关。当{0, CL[6:0]}的值小于{EPCnL, CCAPnL[6:0]}时，输出为低；当{0, CL[6:0]}的值等于或大于{EPCnL, CCAPnL[6:0]}时，输出为高。当CL的值由7F变为00溢出时，{EPCnH, CCAPnH[6:0]}的内容装载到{EPCnL, CCAPnL[6:0]}中。这样就可实现无干扰地更新PWM。要使能PWM模式，模块CCAPMn寄存器的PWMn和ECOMn位必须置位。

当PWM是7位的时： $\text{PWM的频率} = \dfrac{\text{PCA时钟输入源频率}}{128}$

PCA时钟输入源可以从以下8种中选择一种：SYSclk， SYSclk/2， SYSclk/4，SYSclk/6，SYSclk/8，SYSclk/12, 定时器0的溢出，ECI/P1.2输入。

举例：设PCA模块工作于7位PWM模式。要求PWM输出频率为38KHz，选SYSclk为PCA/PWM时钟输入源，求出SYSclk的值。

由计算公式38000=SYSclk/128 ，得到外部时钟频率SYSclk=38000 x 128 x 1=4,864,000

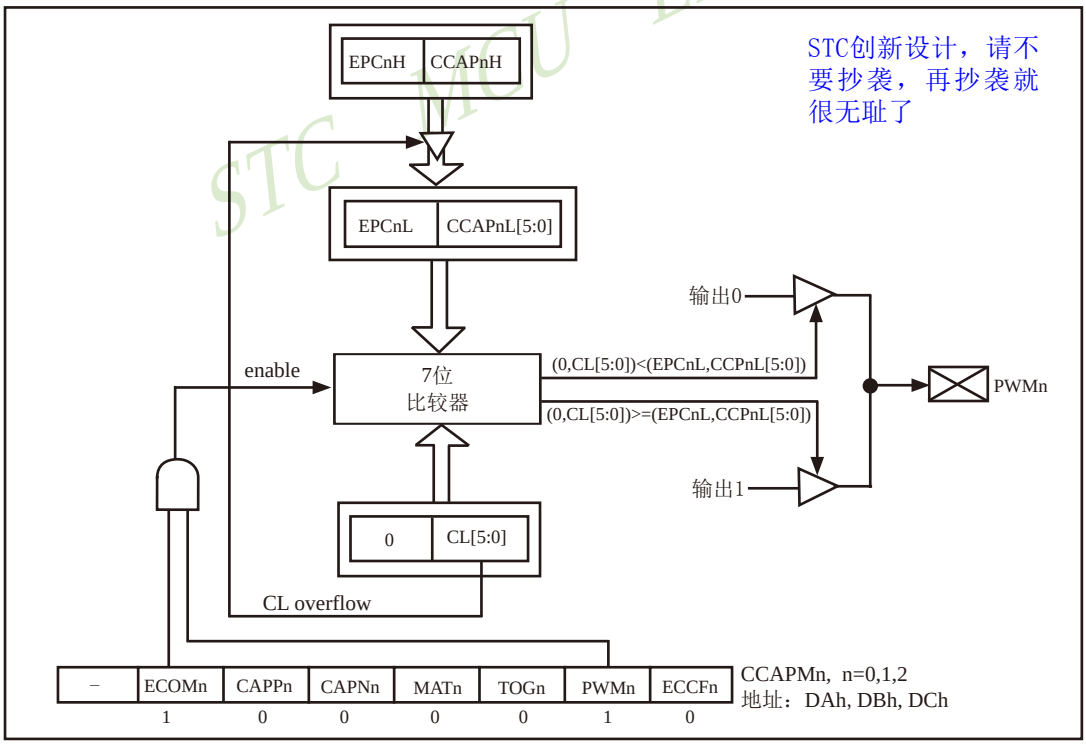
如果要实现可调频率的PWM输出, 可选择定时器0的溢出率或者ECI脚的输入作为PCA/PWM的时钟输入源

当EPCnL = 0及CCAPnL = 80H时, PWM固定输出高

当EPCnL = 1及CCAPnL = 0FFH时, PWM固定输出低

11.3.4.3 6位脉宽调节模式(PWM) (STC创新设计，请不要抄袭)

当[EBSn_1,EBSn_0]=[1,0]时，PCA模块n工作于6位PWM模式，此时将{0, CL[5:0]}与捕获寄存器[EPCnL, CCAPnL[5:0]]进行比较。PWM模式的结构如下图所示。



PCA PWM mode / 可调制脉冲宽度输出模式结构图(PCA模块工作于6位PWM模式)

当PCA模块工作于6位PWM模式时，由于所有模块共用仅有的PCA定时器，所有它们的输出频率相同。各个模块的输出占空比是独立变化的，与使用的捕获寄存器{EPCnL, CCAPnL[5:0]}有关。当{0, CL[5:0]}的值小于{EPCnL, CCAPnL[5:0]}时，输出为低；当{0, CL[5:0]}的值等于或大于{EPCnL, CCAPnL[5:0]}时，输出为高。当CL的值由3F变为00溢出时，{EPCnH, CCAPnH[5:0]}的内容装载到{EPCnL, CCAPnL[5:0]}中。这样就可实现无干扰地更新PWM。要使能PWM模式，模块CCAPMn寄存器的PWMn和ECOMn位必须置位。

当PWM是6位的时：　PWM的频率= $\frac{\text{PCA时钟输入源频率}}{64}$

PCA时钟输入源可以从以下8种中选择一种：SYSclk，　SYSclk/2，　SYSclk/4，SYSclk/6，SYSclk/8，SYSclk/12, 定时器0的溢出，ECI/P1.2输入。

举例：设PCA模块工作于6位PWM模式。要求PWM输出频率为38KHz，选SYSclk为PCA/PWM时钟输入源，求出SYSclk的值。

由计算公式38000=SYSclk/64 ，得到外部时钟频率SYSclk=38000 x 64 x 1=2,432,000

如果要实现可调频率的PWM输出, 可选择定时器0的溢出率或者ECI脚的输入作为PCA/PWM的时钟输入源

当EPCnL = 0及CCAPnL = 0C0H时, PWM固定输出高

当EPCnL = 1及CCAPnL = 0FFH时, PWM固定输出低

STC MCU Limited

11.4 用CCP/PCA功能扩展外部中断的测试程序(C和汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------*/
/* --- 演示  STC 1T 系列单片机  用PCA功能扩展外部中断 ---------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

sfr     P_SW1   =   0xA2;               //外设功能切换寄存器1

#define CCP_S0  0x10                    //P_SW1.4
#define CCP_S1  0x20                    //P_SW1.5

sfr     CCON    =   0xD8;               //PCA控制寄存器
sbit    CCF0    =   CCON^0;             //PCA模块0中断标志
sbit    CCF1    =   CCON^1;             //PCA模块1中断标志
sbit    CR      =   CCON^6;             //PCA定时器运行控制位
sbit    CF      =   CCON^7;             //PCA定时器溢出标志
sfr     CMOD    =   0xD9;               //PCA模式寄存器
sfr     CL      =   0xE9;               //PCA定时器低字节
sfr     CH      =   0xF9;               //PCA定时器高字节
sfr     CCAPM0      =   0xDA;           //PCA模块0模式寄存器
sfr     CCAP0L      =   0xEA;           //PCA模块0捕获寄存器 LOW
sfr     CCAP0H      =   0xFA;           //PCA模块0捕获寄存器 HIGH
sfr     CCAPM1      =   0xDB;           //PCA模块1模式寄存器
sfr     CCAP1L      =   0xEB;           //PCA模块1捕获寄存器 LOW
sfr     CCAP1H      =   0xFB;           //PCA模块1捕获寄存器 HIGH
sfr     PCAPWM0     =   0xf2;
```

STC MCU Limited

```
sfr     PCAPWM1     =  0xf3;

sbit    PCA_LED     =  P1^0;                    //PCA测试LED

void PCA_isr() interrupt 7 using 1
{
        CCF0 = 0;                               //清中断标志
        PCA_LED = !PCA_LED;                     //测试LED取反
}
void main()
{
        ACC     =       P_SW1;
        ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=0
        P_SW1   =       ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      ACC     =       P_SW1;
//      ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=1 CCP_S1=0
//      ACC     |=      CCP_S0;         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      P_SW1   =       ACC;
//
//      ACC     =       P_SW1;
//      ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=1
//      ACC     |=      CCP_S1;         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      P_SW1   =       ACC;

        CCON = 0;                               //初始化PCA控制寄存器
                                                //PCA定时器停止
                                                //清除CF标志
                                                //清除模块中断标志
        CL = 0;                                 //复位PCA寄存器
        CH = 0;
        CMOD = 0x00;                            //设置PCA时钟源
                                                //禁止PCA定时器溢出中断
        CCAPM0 = 0x11;                          //PCA模块0为下降沿触发
//      CCAPM0 = 0x21;                          //PCA模块0为上升沿沿触发
//      CCAPM0 = 0x31;                          //PCA模块0为上升沿/下降沿触发

        CR = 1;                                 //PCA定时器开始工作
        EA = 1;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ------------------------------------------------------------*/
/* --- 演示  STC 1T 系列单片机  用PCA功能扩展外部中断 ----------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

P_SW1       EQU     0A2H            ;外设功能切换寄存器1

CCP_S0      EQU     10H             ;P_SW1.4
CCP_S1      EQU     20H             ;P_SW1.5

CCON        EQU     0D8H            ;PCA控制寄存器
CCF0        BIT     CCON.0          ;PCA模块0中断标志
CCF1        BIT     CCON.1          ;PCA模块1中断标志
CR          BIT     CCON.6          ;PCA定时器运行控制位
CF          BIT     CCON.7          ;PCA定时器溢出标志
CMOD        EQU     0D9H            ;PCA模式寄存器
CL          EQU     0E9H            ;PCA定时器低字节
CH          EQU     0F9H            ;PCA定时器高字节
CCAPM0      EQU     0DAH            ;PCA模块0模式寄存器
CCAP0L      EQU     0EAH            ;PCA模块0捕获寄存器 LOW
CCAP0H      EQU     0FAH            ;PCA模块0捕获寄存器 HIGH
CCAPM1      EQU     0DBH            ;PCA模块1模式寄存器
CCAP1L      EQU     0EBH            ;PCA模块1捕获寄存器 LOW
CCAP1H      EQU     0FBH            ;PCA模块1捕获寄存器 HIGH

PCA_LED     BIT     P1.0            ;PCA测试LED

;--------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     003BH
PCA_ISR:
        CLR     CCF0                ;清中断标志
        CPL     PCA_LED             ;测试LED取反
        RETI
;--------------------------------------
```

STC MCU Limited

```
        ORG     0100H
MAIN:
        MOV     A,      P_SW1
        ANL     A,      #0CFH           //CCP_S0=0 CCP_S1=0
        MOV     P_SW1,  A               //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      MOV     A,      P_SW1
//      ANL     A,      #0CFH           //CCP_S0=1 CCP_S1=0
//      ORL     A,      #CCP_S0         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      MOV     P_SW1,  A
//
//      MOV     A,      P_SW1
//      ANL     A,      #0CFH           //CCP_S0=0 CCP_S1=1
//      ORL     A,      #CCP_S1         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      MOV     P_SW1,  A

        MOV     CCON,   #0              ;初始化PCA控制寄存器
                                        ;PCA定时器停止
                                        ;清除CF标志
                                        ;清除模块中断标志
        CLR     A                       ;
        MOV     CL,     A               ;复位PCA寄存器
        MOV     CH,     A               ;
        MOV     CMOD,   #00H            ;设置PCA时钟源
                                        ;禁止PCA定时器溢出中断
        MOV     CCAPM0,         #11H    ;PCA模块0捕获CCP0(P1.3)的下降沿
;       MOV     CCAPM0,         #21H    ;PCA模块0捕获CCP0(P1.3)的上升沿
;       MOV     CCAPM0,         #31H    ;PCA模块0捕获CCP0(P1.3)的上升沿/下降沿
;-----------------------------
        SETB    CR                      ;PCA定时器开始工作
        SETB    EA

        SJMP    $
;---------------------------------------
        END
```

STC MCU Limited

11.5 用CCP/PCA功能实现16位定时器的测试程序(C和汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------*/
/* --- 演示  STC 1T 系列单片机  用PCA功能实现16位定时器 ----------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ----------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC    18432000L
#define   T100Hz  (FOSC / 12 / 100)

typedef unsigned char   BYTE;
typedef unsigned int    WORD;

sfr     P_SW1   =   0xA2;                   //外设功能切换寄存器1

#define CCP_S0  0x10                        //P_SW1.4
#define CCP_S1  0x20                        //P_SW1.5

sfr     CCON    =   0xD8;                   //PCA控制寄存器
sbit    CCF0    =   CCON^0;                 //PCA模块0中断标志
sbit    CCF1    =   CCON^1;                 //PCA模块1中断标志
sbit    CR      =   CCON^6;                 //PCA定时器运行控制位
sbit    CF      =   CCON^7;                 //PCA定时器溢出标志
sfr     CMOD    =   0xD9;                   //PCA模式寄存器
sfr     CL      =   0xE9;                   //PCA定时器低字节
sfr     CH      =   0xF9;                   //PCA定时器高字节
sfr     CCAPM0  =   0xDA;                   //PCA模块0模式寄存器
sfr     CCAP0L  =   0xEA;                   //PCA模块0捕获寄存器 LOW
sfr     CCAP0H  =   0xFA;                   //PCA模块0捕获寄存器 HIGH
sfr     CCAPM1  =   0xDB;                   //PCA模块1模式寄存器
sfr     CCAP1L  =   0xEB;                   //PCA模块1捕获寄存器 LOW
sfr     CCAP1H  =   0xFB;                   //PCA模块1捕获寄存器 HIGH
sfr     PCAPWM0 =   0xf2;
sfr     PCAPWM1 =   0xf3;

sbit    PCA_LED =   P1^0;                   //PCA测试LED
BYTE cnt;
WORD value;
```

STC MCU Limited

```
void PCA_isr() interrupt 7 using 1
{
	CCF0 = 0;                                       //清中断标志
	CCAP0L = value;
	CCAP0H = value >> 8;                            //更新比较值
	value += T100Hz;
	if (cnt-- == 0)
	{
		cnt = 100;                              //记数100次
		PCA_LED = !PCA_LED;                     //每秒闪烁一次
	}
}

void main()
{
	ACC	=	P_SW1;
	ACC	&=	~(CCP_S0 | CCP_S1);             //CCP_S0=0 CCP_S1=0
	P_SW1	=	ACC;                            //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//	ACC	=	P_SW1;
//	ACC	&=	 ~(CCP_S0 | CCP_S1);            //CCP_S0=1 CCP_S1=0
//	ACC	|=	CCP_S0;                 //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//	P_SW1	=	ACC;
//
//	ACC	=	P_SW1;
//	ACC	&=	~(CCP_S0 | CCP_S1);             //CCP_S0=0 CCP_S1=1
//	ACC	|=	CCP_S1;                 //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//	P_SW1	=	ACC;

	CCON	=	0;                              //初始化PCA控制寄存器
                                                        //PCA定时器停止
                                                        //清除CF标志
                                                        //清除模块中断标志
	CL	=	0;                              //复位PCA寄存器
	CH	=	0;
	CMOD	=	0x00;                           //设置PCA时钟源
                                                        //禁止PCA定时器溢出中断
	value = T100Hz;
	CCAP0L = value;
	CCAP0H = value >> 8;                            //初始化PCA模块0
	 value += T100Hz;
	CCAPM0 = 0x49;                                  //PCA模块0为16位定时器模式

	CR = 1;                                         //PCA定时器开始工作
	EA = 1;
	cnt = 0;

	while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ---------------------------------------------------------*/
/* --- 演示  STC 1T 系列单片机  用PCA功能实现16位定时器 ----------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ---------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

T100Hz          EQU     3C00H           ;(18432000 / 12 / 100)
P_SW1           EQU     0A2H            ;外设功能切换寄存器1

CCP_S0          EQU     10H             ;P_SW1.4
CCP_S1          EQU     20H             ;P_SW1.5

CCON            EQU     0D8H            ;PCA控制寄存器
CCF0            BIT     CCON.0          ;PCA模块0中断标志
CCF1            BIT     CCON.1          ;PCA模块1中断标志
CR              BIT     CCON.6          ;PCA定时器运行控制位
CF              BIT     CCON.7          ;PCA定时器溢出标志
CMOD            EQU     0D9H            ;PCA模式寄存器
CL              EQU     0E9H            ;PCA定时器低字节
CH              EQU     0F9H            ;PCA定时器高字节
CCAPM0          EQU     0DAH            ;PCA模块0模式寄存器
CCAP0L          EQU     0EAH            ;PCA模块0捕获寄存器 LOW
CCAP0H          EQU     0FAH            ;PCA模块0捕获寄存器 HIGH
CCAPM1          EQU     0DBH            ;PCA模块1模式寄存器
CCAP1L          EQU     0EBH            ;PCA模块1捕获寄存器 LOW
CCAP1H          EQU     0FBH            ;PCA模块1捕获寄存器 HIGH

PCA_LED         BIT     P1.0            ;PCA测试LED

CNT             EQU     20H
;--------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     003BH
        LJMP    PCA_ISR
;--------------------------------------
        ORG     0100H
MAIN:
        MOV     SP,     #3FH
        MOV     A,      P_SW1
        ANL     A,      #0CFH           //CCP_S0=0 CCP_S1=0
        MOV     P_SW1,  A               //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)
```

STC MCU Limited

```
//      MOV     A,      P_SW1
//      ANL     A,      #0CFH                   //CCP_S0=1 CCP_S1=0
//      ORL     A,      #CCP_S0         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      MOV     P_SW1,  A
//
//      MOV     A,      P_SW1
//      ANL     A,      #0CFH                   //CCP_S0=0 CCP_S1=1
//      ORL     A,      #CCP_S1         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      MOV     P_SW1,  A

        MOV     CCON,   #0                              ;初始化PCA控制寄存器
                                                        ;PCA定时器停止
                                                        ;清除CF标志
                                                        ;清除模块中断标志
        CLR     A                                       ;
        MOV     CL,     A                               ;复位PCA寄存器
        MOV     CH,     A                               ;
        MOV     CMOD,   #00H                            ;设置PCA时钟源
                                                        ;禁止PCA定时器溢出中断
;------------------------------
        MOV     CCAP0L,         #LOW T100Hz             ;
        MOV     CCAP0H,         #HIGH T100Hz            ;初始化PCA模块0
        MOV     CCAPM0,         #49H                    ;PCA模块0为16位定时器模式
;------------------------------
        SETB    CR                                      ;PCA定时器开始工作
        SETB    EA
        MOV     CNT,    #100

        SJMP    $
;----------------------------------------
PCA_ISR:
        PUSH    PSW
        PUSH    ACC
        CLR     CCF0                                    ;清中断标志
        MOV     A,      CCAP0L
        ADD     A,      #LOW T100Hz                     ;更新比较值
        MOV     CCAP0L,A
        MOV     A,      CCAP0H
        ADDC    A,      #HIGH T100Hz
        MOV     CCAP0H,         A
        DJNZ    CNT,    PCA_ISR_EXIT                    ;记数100次
        MOV     CNT,    #100
        CPL     PCA_LED                                 ;每秒闪烁一次
PCA_ISR_EXIT:
        POP     ACC
        POP     PSW
        RETI
;----------------------------------------
        END
```

STC MCU Limited

11.6 CCP/PCA输出高速脉冲的测试程序(C和汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------*/
/* ---演示  STC 1T 系列单片机  PCA输出高速脉冲  ----------------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC    18432000L
#define T100KHz (FOSC / 4 / 100000)

typedef unsigned char BYTE;
typedef unsigned int WORD;

sfr     P_SW1   =   0xA2;                   //外设功能切换寄存器1

#define CCP_S0          0x10                //P_SW1.4
#define CCP_S1          0x20                //P_SW1.5

sfr     CCON    =   0xD8;                   //PCA控制寄存器
sbit    CCF0    =   CCON^0;                 //PCA模块0中断标志
sbit    CCF1    =   CCON^1;                 //PCA模块1中断标志
sbit    CR      =   CCON^6;                 //PCA定时器运行控制位
sbit    CF      =   CCON^7;                 //PCA定时器溢出标志
sfr     CMOD    =   0xD9;                   //PCA模式寄存器
sfr     CL      =   0xE9;                   //PCA定时器低字节
sfr     CH      =   0xF9;                   //PCA定时器高字节
sfr     CCAPM0  =   0xDA;                   //PCA模块0模式寄存器
sfr     CCAP0L  =   0xEA;                   //PCA模块0捕获寄存器 LOW
sfr     CCAP0H  =   0xFA;                   //PCA模块0捕获寄存器 HIGH
sfr     CCAPM1  =   0xDB;                   //PCA模块1模式寄存器
sfr     CCAP1L  =   0xEB;                   //PCA模块1捕获寄存器 LOW
sfr     CCAP1H  =   0xFB;                   //PCA模块1捕获寄存器 HIGH
sfr     PCAPWM0 =   0xf2;
sfr     PCAPWM1 =   0xf3;
```

STC MCU Limited

```
sbit    PCA_LED  =  P1^0;                    //PCA测试LED

BYTE cnt;
WORD value;

void PCA_isr() interrupt 7 using 1
{
        CCF0 = 0;                           //清中断标志
        CCAP0L = value;
        CCAP0H = value >> 8;                //更新比较值
        value += T100KHz;
}

void main()
{
        ACC     =       P_SW1;
        ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=0
        P_SW1   =       ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      ACC     =       P_SW1;
//      ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=1 CCP_S1=0
//      ACC     |=      CCP_S0;         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      P_SW1   =       ACC;
//
//      ACC     =       P_SW1;
//      ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=1
//      ACC     |=      CCP_S1;         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      P_SW1   =       ACC;

        CCON    =       0;                      //初始化PCA控制寄存器
                                                //PCA定时器停止
                                                //清除CF标志
                                                //清除模块中断标志
        CL      =       0;                      //复位PCA寄存器
        CH      =       0;
        CMOD    =       0x02;                   //设置PCA时钟源
                                                //禁止PCA定时器溢出中断
        value   =       T100KHz;
        CCAP0L  =       value;                  //P1.1输出100KHz方波
        CCAP0H  =       value >> 8;             //初始化PCA模块0
        value   +=      T100KHz;
        CCAPM0  =       0x4d;           //PCA模块0为16位定时器模式,同时反转CCP0(P1.1)口

        CR      =       1;              //PCA定时器开始工作
        EA      =       1;
        cnt     =       0;

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------*/
/* ---演示  STC 1T 系列单片机  PCA输出高速脉冲 ----------------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ----------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序，  ----------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 ----------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

T100KHz         EQU     2EH                     ;(18432000 / 4 / 100000)

P_SW1           EQU     0A2H                    ;外设功能切换寄存器1

CCP_S0          EQU     10H                     ;P_SW1.4
CCP_S1          EQU     20H                     ;P_SW1.5

CCON            EQU     0D8H                    ;PCA控制寄存器
CCF0            BIT     CCON.0                  ;PCA模块0中断标志
CCF1            BIT     CCON.1                  ;PCA模块1中断标志
CR              BIT     CCON.6                  ;PCA定时器运行控制位
CF              BIT     CCON.7                  ;PCA定时器溢出标志
CMOD            EQU     0D9H                    ;PCA模式寄存器
CL              EQU     0E9H                    ;PCA定时器低字节
CH              EQU     0F9H                    ;PCA定时器高字节
CCAPM0          EQU     0DAH                    ;PCA模块0模式寄存器
CCAP0L          EQU     0EAH                    ;PCA模块0捕获寄存器 LOW
CCAP0H          EQU     0FAH                    ;PCA模块0捕获寄存器 HIGH
CCAPM1          EQU     0DBH                    ;PCA模块1模式寄存器
CCAP1L          EQU     0EBH                    ;PCA模块1捕获寄存器 LOW
CCAP1H          EQU     0FBH                    ;PCA模块1捕获寄存器 HIGH

;------------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     003BH
PCA_ISR:
        PUSH    PSW
        PUSH    ACC
        CLR     CCF0                            ;清中断标志
        MOV     A,      CCAP0L
        ADD     A,      #T100KHz
        MOV     CCAP0L,         A
```

STC MCU Limited

```
        CLR     A
        ADDC    A,      CCAP0H
        MOV     CCAP0H,         A
PCA_ISR_EXIT:
        POP     ACC
        POP     PSW
        RETI
;----------------------------------------
        ORG     0100H
MAIN:
        MOV     A,      P_SW1
        ANL     A,      #0CFH                   //CCP_S0=0 CCP_S1=0
        MOV     P_SW1,  A                       //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      MOV     A,      P_SW1
//      ANL     A,      #0CFH                   //CCP_S0=1 CCP_S1=0
//      ORL     A,      #CCP_S0         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      MOV     P_SW1,  A
//
//      MOV     A,      P_SW1
//      ANL     A,      #0CFH                   //CCP_S0=0 CCP_S1=1
//      ORL     A,      #CCP_S1         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      MOV     P_SW1,  A

        MOV     CCON,   #0                      ;初始化PCA控制寄存器
                                                ;PCA定时器停止
                                                ;清除CF标志
                                                ;清除模块中断标志
        CLR     A                               ;
        MOV     CL,     A                       ;复位PCA寄存器
        MOV     CH,     A                               ;
        MOV     CMOD,   #02H                    ;设置PCA时钟源
                                                ;禁止PCA定时器溢出中断
;------------------------------
        MOV     CCAP0L,         #T100KHz        ;P1.3输出100KHz方波
        MOV     CCAP0H,         #0              ;初始化PCA模块0
        MOV     CCAPM0,         #4dH            ;PCA模块0为16位定时模式并使能PCA中断
;------------------------------
        SETB    CR                              ;PCA定时器开始工作
        SETB    EA

        SJMP    $
;----------------------------------------
        END
```

STC MCU Limited

11.7 CCP/PCA输出PWM(6位+7位+8位)的测试程序(C和汇编)

1. C程序:

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------*/
/* ---STC15F2K60S2 系列 PCA输出6/7/8位PWM举例---------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ---------------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC    18432000L

typedef unsigned char BYTE;
typedef unsigned int WORD;

sfr     P_SW1   = 0xA2;                 //外设功能切换寄存器1

#define CCP_S0  0x10                    //P_SW1.4
#define CCP_S1  0x20                    //P_SW1.5

sfr     CCON    =       0xD8;           //PCA控制寄存器
sbit    CCF0    =       CCON^0;         //PCA模块0中断标志
sbit    CCF1    =       CCON^1;         //PCA模块1中断标志
sbit    CR      =        CCON^6;        //PCA定时器运行控制位
sbit    CF      =       CCON^7;         //PCA定时器溢出标志
sfr     CMOD    =       0xD9;           //PCA模式寄存器
sfr     CL      =       0xE9;           //PCA定时器低字节
sfr     CH      =       0xF9;           //PCA定时器高字节
sfr     CCAPM0=         0xDA;           //PCA模块0模式寄存器
sfr     CCAP0L =        0xEA;           //PCA模块0捕获寄存器 LOW
sfr     CCAP0H =        0xFA;           //PCA模块0捕获寄存器 HIGH
sfr     CCAPM1=         0xDB;           //PCA模块1模式寄存器
sfr     CCAP1L =        0xEB;           //PCA模块1捕获寄存器 LOW
sfr     CCAP1H =        0xFB;           //PCA模块1捕获寄存器 HIGH
```

STC MCU Limited

```
sfr     CCAPM2      =   0xDC;       //PCA模块2模式寄存器
sfr     CCAP2L      =   0xEC;       //PCA模块2捕获寄存器 LOW
sfr     CCAP2H      =   0xFC;       //PCA模块2捕获寄存器 HIGH
sfr     PCA_PWM0    =   0xf2;       //PCA模块0的PWM寄存器
sfr     PCA_PWM1    =   0xf3;       //PCA模块1的PWM寄存器
sfr     PCA_PWM2    =   0xf4;       //PCA模块2的PWM寄存器

void main()
{
        ACC     =   P_SW1;
        ACC     &=  ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=0
        P_SW1   =   ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      ACC     =   P_SW1;
//      ACC     &=  ~(CCP_S0 | CCP_S1);     //CCP_S0=1 CCP_S1=0
//      ACC     |=  CCP_S0;          //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      P_SW1   =   ACC;
//
//      ACC     =   P_SW1;
//      ACC     &=  ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=1
//      ACC     |=  CCP_S1;          //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      P_SW1   =   ACC;

        CCON    =   0;                      //初始化PCA控制寄存器
                                            //PCA定时器停止
                                            //清除CF标志
                                            //清除模块中断标志
        CL      =   0;                      //复位PCA寄存器
        CH      =   0;
        CMOD    =   0x02;                   //设置PCA时钟源
                                            //禁止PCA定时器溢出中断
        PCA_PWM0    =   0x00;               //PCA模块0工作于8位PWM
        CCAP0H  =  CCAP0L  =  0x20;         //PWM0的占空比为87.5% ((100H-20H)/100H)
        CCAPM0      =   0x42;               //PCA模块0为8位PWM模式

        PCA_PWM1 = 0x40;                    //PCA模块1工作于7位PWM
        CCAP1H = CCAP1L = 0x20;             //PWM1的占空比为75% ((80H-20H)/80H)
        CCAPM1 = 0x42;                      //PCA模块1为7位PWM模式

        PCA_PWM2 = 0x80;                    //PCA模块2工作于6位PWM
        CCAP2H = CCAP2L = 0x20;             //PWM2的占空比为50% ((40H-20H)/40H)
        CCAPM2 = 0x42;                      //PCA模块2为6位PWM模式

        CR = 1;                             //PCA定时器开始工作

        while (1);
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ------------------------------------------------------------*/
/* ---STC15F2K60S2 系列 PCA输出6/7/8位PWM举例---------------------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -----------------------------------------------------------*/
/* --- Web: www.STCMCU.com ------------------------------------------------------------*/
/* 如果要在程序中使用或在文章中引用该程序， ---------------------------------*/
/* 请在程序中或文章中注明使用了STC的资料及程序 -----------------------------------*/
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可--------*/
/*---------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

P_SW1           EQU     0A2H                    ;外设功能切换寄存器1

CCP_S0          EQU     10H                     ;P_SW1.4
CCP_S1          EQU     20H                     ;P_SW1.5

CCON            EQU     0D8H                    ;PCA控制寄存器
CCF0            BIT     CCON.0                  ;PCA模块0中断标志
CCF1            BIT     CCON.1                  ;PCA模块1中断标志
CR              BIT     CCON.6                  ;PCA定时器运行控制位
CF              BIT     CCON.7                  ;PCA定时器溢出标志
CMOD            EQU     0D9H                    ;PCA模式寄存器
CL              EQU     0E9H                    ;PCA定时器低字节
CH              EQU     0F9H                    ;PCA定时器高字节
CCAPM0          EQU     0DAH                    ;PCA模块0模式寄存器
CCAP0L          EQU     0EAH                    ;PCA模块0捕获寄存器 LOW
CCAP0H          EQU     0FAH                    ;PCA模块0捕获寄存器 HIGH
CCAPM1          EQU     0DBH                    ;PCA模块1模式寄存器
CCAP1L          EQU     0EBH                    ;PCA模块1捕获寄存器 LOW
CCAP1H          EQU     0FBH                    ;PCA模块1捕获寄存器 HIGH
CCAPM2          EQU     0DCH                    ;PCA模块2模式寄存器
CCAP2L          EQU     0ECH                    ;PCA模块2捕获寄存器 LOW
CCAP2H          EQU     0FCH                    ;PCA模块2捕获寄存器 HIGH
PCA_PWM0        EQU     0F2H                    ;PCA模块0的PWM寄存器
PCA_PWM1        EQU     0F3H                    ;PCA模块1的PWM寄存器
PCA_PWM2        EQU     0F4H                    ;PCA模块2的PWM寄存器
;-----------------------------------------
        ORG     0000H
        LJMP    MAIN
;-----------------------------------------
        ORG     0100H
MAIN:
        MOV     A,      P_SW1
        ANL     A,      #0CFH                   //CCP_S0=0 CCP_S1=0
```

STC MCU Limited

```
        MOV     P_SW1,  A               //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      MOV     A,      P_SW1
//      ANL     A,      #0CFH           //CCP_S0=1 CCP_S1=0
//      ORL     A,      #CCP_S0         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      MOV     P_SW1,  A
//
//      MOV     A,      P_SW1
//      ANL     A,      #0CFH           //CCP_S0=0 CCP_S1=1
//      ORL     A,      #CCP_S1         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      MOV     P_SW1,  A

        MOV     CCON,   #0                      ;初始化PCA控制寄存器
                                                ;PCA定时器停止
                                                ;清除CF标志
                                                ;清除模块中断标志
        CLR     A                               ;
        MOV     CL,     A                       ;复位PCA计数器
        MOV     CH,     A                       ;
        MOV     CMOD,   #02H                    ;设置PCA时钟源
                                                ;禁止PCA定时器溢出中断
;------------------------------
        MOV     PCA_PWM0,       #00H            ;PCA模块0工作于8位PWM
        MOV     A,              #020H           ;
        MOV     CCAP0H,         A               ;PWM0的占空比为87.5% ((100H-20H)/100H)
        MOV     CCAP0L,         A               ;
        MOV     CCAPM0,         #42H            ;PCA模块0为8位PWM模式
;------------------------------
        MOV     PCA_PWM1,       #40H            ;PCA模块1工作于7位PWM
        MOV     A,              #020H           ;
        MOV     CCAP1H,         A               ;PWM1的占空比为75% ((80H-20H)/80H)
        MOV     CCAP1L,         A               ;
        MOV     CCAPM1,         #42H            ;PCA模块1为7位PWM模式
;------------------------------
        MOV     PCA_PWM2,       #80H            ;PCA模块2工作于6位PWM
        MOV     A,              #020H           ;
        MOV     CCAP2H,         A               ;PWM2的占空比为50% ((40H-20H)/40H)
        MOV     CCAP2L,         A               ;
        MOV     CCAPM2,         #42H            ;PCA模块2为6位PWM模式
;------------------------------
        SETB    CR                              ;PCA定时器开始工作

        SJMP    $
;----------------------------------------
        END
```

STC MCU Limited

11.8 用CCP/PCA高速脉冲输出功能实现3路9~16位PWM的程序——每通道占用系统时间小于0.6%

下文为利用CCP/PCA高速脉冲输出功能实现3路9~16位PWM（每通道占用系统时间小于0.6%）的测试程序（包括C语言程序和汇编程序）。

1、C语言程序

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ------------------------------------------------------------------------------*/
/* --- Mobile: (86)13922805190 ----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------------------------------*/
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/

#include  <reg52.h>

/*************    功能说明    **************

输出3路9~16位PWM信号。

PWM频率 = MAIN_Fosc / PWM_DUTY,
假设 MAIN_Fosc = 24MHZ, PWM_DUTY = 6000, 则输出PWM频率为4000HZ.

*****************************************/
/*****************************用户宏定义*****************************************************/
#define    MAIN_Fosc        24000000UL      //定义主时钟

#define    PWM_DUTY         6000      //定义PWM的周期，数值为PCA所选择的时钟脉冲个数。
#define    PWM_HIGH_MIN 80
                                     //限制PWM输出的最小占空比, 避免中断里重装参数时间不够。
#define    PWM_HIGH_MAX             (PWM_DUTY - PWM_HIGH_MIN)
                                     //限制PWM输出的最大占空比。

/*********************************************************************************************/
#define    PCA0                     0
#define    PCA1                     1
#define    PCA2                     2
#define    PCA_Counter              3
#define    PCA_P12_P11_P10_P37      (0<<4)
#define    PCA_P34_P35_P36_P37      (1<<4)
#define    PCA_P24_P25_P26_P27      (2<<4)
#define    PCA_Mode_PWM             0x42
#define    PCA_Mode_Capture         0
#define    PCA_Mode_SoftTimer       0x48
#define    PCA_Mode_HighPulseOutput   0x4c
```

STC MCU Limited

```
#define  PCA_Clock_1T            (4<<1)
#define  PCA_Clock_2T            (1<<1)
#define  PCA_Clock_4T            (5<<1)
#define  PCA_Clock_6T            (6<<1)
#define  PCA_Clock_8T            (7<<1)
#define  PCA_Clock_12T           (0<<1)
#define  PCA_Clock_Timer0_OF     (2<<1)
#define  PCA_Clock_ECI           (3<<1)
#define  PCA_Rise_Active         (1<<5)
#define  PCA_Fall_Active         (1<<4)
#define  PCA_PWM_8bit            (0<<6)
#define  PCA_PWM_7bit            (1<<6)
#define  PCA_PWM_6bit            (2<<6)

#define  ENABLE                  1
#define  DISABLE                 0

typedef  unsigned char           u8;
typedef  unsigned int            u16;
typedef  unsigned long           u32;

sfr      AUXR1       =       0xA2;
sfr      CCON        =       0xD8;
sfr      CMOD        =       0xD9;
sfr      CCAPM0      =       0xDA;       //PCA模块0的工作模式寄存器。
sfr      CCAPM1      =       0xDB;       //PCA模块1的工作模式寄存器。
sfr      CCAPM2      =       0xDC;       //PCA模块2的工作模式寄存器。

sfr      CL          =       0xE9;
sfr      CCAP0L      =       0xEA;       //PCA模块0的捕捉/比较寄存器低8位。
sfr      CCAP1L      =       0xEB;       //PCA模块1的捕捉/比较寄存器低8位。
sfr      CCAP2L      =       0xEC;       //PCA模块2的捕捉/比较寄存器低8位。

sfr      CH          =       0xF9;
sfr      CCAP0H      =       0xFA;       //PCA模块0的捕捉/比较寄存器高8位。
sfr      CCAP1H      =       0xFB;       //PCA模块1的捕捉/比较寄存器高8位。
sfr      CCAP2H      =       0xFC;       //PCA模块2的捕捉/比较寄存器高8位。

sbit     CCF0        =       CCON^0;     //PCA模块0中断标志, 由硬件置位, 必须由软件清0
sbit     CCF1        =       CCON^1;     //PCA模块1中断标志, 由硬件置位, 必须由软件清0
sbit     CCF2        =       CCON^2;     //PCA模块2中断标志, 由硬件置位, 必须由软件清0
sbit     CR          =       CCON^6;     //1: 允许PCA计数器计数，0: 禁止计数。
sbit     CF          =       CCON^7;
                             //PCA计数器溢出（CH，CL由FFFFH变为0000H）标志。
                             //PCA计数器溢出后由硬件置位，必须由软件清0。
sbit     PPCA        =       IP^7;       //PCA 中断 优先级设定位

sfr      P2M1        =       0x95;       //P2M1.n,P2M0.n   =00--->Standard,   01--->push-pull
sfr      P2M0        =       0x96;       //                =10--->pure input, 11--->open drain
```

STC MCU Limited

```
//=========================================================================
sbit    P25 = P2^5;
sbit    P26 = P2^6;
sbit    P27 = P2^7;

u16     CCAP0_tmp,PWM0_high,PWM0_low;
u16     CCAP1_tmp,PWM1_high,PWM1_low;
u16     CCAP2_tmp,PWM2_high,PWM2_low;
u16     pwm0,pwm1,pwm2;
void    PWMn_Update(u8 PCA_id, u16 pwm);
void    PCA_Init(void);
void    delay_ms(u8 ms);

/********************* 主函数 **************************/
void main(void)
{

    PCA_Init();                         //PCA初始化
    EA = 1;
    P2M1 &= ~(0xe0);                    //P2.7 P2.6 P2.5 设置为推挽输出
    P2M0 |=  (0xe0);

    while (1)
    {
        delay_ms(2);

        if(++pwm0 >= PWM_HIGH_MAX)  pwm0 = PWM_HIGH_MIN;
        PWMn_Update(PCA0,pwm0);

        if(++pwm1 >= PWM_HIGH_MAX)  pwm1 = PWM_HIGH_MIN;
        PWMn_Update(PCA1,pwm1);

        if(++pwm2 >= PWM_HIGH_MAX)  pwm2 = PWM_HIGH_MIN;
        PWMn_Update(PCA2,pwm2);
    }
}

//========================================================================
// 函数: void  delay_ms(u8 ms)
// 描述: 延时函数。
// 参数: ms,要延时的ms数, 这里只支持1~255ms. 自动适应主时钟.
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//========================================================================
void  delay_ms(u8 ms)
{
    unsigned int i;
```

STC MCU Limited

```
        do
        {
                i = MAIN_Fosc / 13000;
                while(--i) ;
        }while(--ms);
}

//========================================================================
// 函数: void PWMn_SetHighReg(unsigned int high)
// 描述: 更新占空比数据。
// 参数: high:        占空比数据，即PWM输出高电平的PCA时钟脉冲个数。
// 返回: 无
// 版本: VER1.0
// 日期: 2013-5-15
// 备注:
//========================================================================
void PWMn_Update(u8 PCA_id, u16 pwm)
{
        if(pwm > PWM_HIGH_MAX)          pwm = PWM_HIGH_MAX;
                                        //如果写入大于最大占空比数据，强制为最大占空比。
        if(pwm < PWM_HIGH_MIN)          pwm = PWM_HIGH_MIN;
                                        //如果写入小于最小占空比数据，强制为最小占空比。

        if(PCA_id == PCA0)
        {
                CR = 0;                 //停止PCA一会， 一般不会影响PWM。
                PWM0_high = pwm;        //数据在正确范围，则装入占空比寄存器。
                PWM0_low = PWM_DUTY - pwm;
                                        //计算并保存PWM输出低电平的PCA时钟脉冲个数。
                CR = 1;                 //启动PCA。
        }
        else if(PCA_id == PCA1)
        {
                CR = 0;                 //停止PCA。
                PWM1_high = pwm;        //数据在正确范围，则装入占空比寄存器。
                PWM1_low = PWM_DUTY - pwm;
                                        //计算并保存PWM输出低电平的PCA时钟脉冲个数。
                CR = 1;                 //启动PCA。
        }
        else if(PCA_id == PCA2)
        {
                CR = 0;                 //停止PCA。
                PWM2_high = pwm;        //数据在正确范围，则装入占空比寄存器。
                PWM2_low = PWM_DUTY - pwm;
                                        //计算并保存PWM输出低电平的PCA时钟脉冲个数。
                CR = 1;                                         //启动PCA。
        }
}
```

STC MCU Limited

```
//========================================================================
// 函数: void        PCA_Init(void)
// 描述: PCA初始化程序.
// 参数: none
// 返回: none.
// 版本: V1.0, 2013-11-22
//========================================================================
void    PCA_Init(void)
{
    CR = 0;
    AUXR1 = (AUXR1 & ~(3<<4)) | PCA_P24_P25_P26_P27;        //切换IO口
    CCAPM0 = (PCA_Mode_HighPulseOutput | ENABLE);
                                                    //16位软件定时、高速脉冲输出、中断模式
    CCAPM1 = (PCA_Mode_HighPulseOutput | ENABLE);
    CCAPM2 = (PCA_Mode_HighPulseOutput | ENABLE);

    CH = 0;
    CL = 0;
    CMOD  = (CMOD  & ~(7<<1)) | PCA_Clock_1T;               //选择时钟源
    PPCA  = 1;                                              // 高优先级中断

    pwm0 = (PWM_DUTY / 4 * 1);                              //给PWM一个初值
    pwm1 = (PWM_DUTY / 4 * 2);
    pwm2 = (PWM_DUTY / 4 * 3);

    PWMn_Update(PCA0,pwm0);
    PWMn_Update(PCA1,pwm1);
    PWMn_Update(PCA2,pwm2);

    CR    = 1;                                              // 运行PCA定时器
}
//========================================================================

//========================================================================
// 函数: void        PCA_Handler (void) interrupt 7
// 描述: PCA中断处理程序.
// 参数: None
// 返回: none.
// 版本: V1.0, 2012-11-22
//========================================================================
void    PCA_Handler (void) interrupt 7
{
    if(CCF0)            //PCA模块0中断
    {
        CCF0 = 0;                   //清PCA模块0中断标志
        if(P25)    CCAP0_tmp += PWM0_high;
                                    //输出为高电平，则给影射寄存器装载高电平时间长度
        else       CCAP0_tmp += PWM0_low;
                                    //输出为低电平，则给影射寄存器装载低电平时间长度
```

STC MCU Limited

```
            CCAP0L = (u8)CCAP0_tmp;
                                        //将影射寄存器写入捕获寄存器，先写CCAP0L
            CCAP0H = (u8)(CCAP0_tmp >> 8);  //后写CCAP0H
        }

        if(CCF1)                        //PCA模块1中断
        {
            CCF1 = 0;                   //清PCA模块1中断标志
            if(P26)   CCAP1_tmp += PWM1_high;
                                        //输出为高电平，则给影射寄存器装载高电平时间长度
            else      CCAP1_tmp += PWM1_low;
                                        //输出为低电平，则给影射寄存器装载低电平时间长度
            CCAP1L = (u8)CCAP1_tmp;     //将影射寄存器写入捕获寄存器，先写CCAP0L
            CCAP1H = (u8)(CCAP1_tmp >> 8);  //后写CCAP0H
        }

        if(CCF2) //PCA模块2中断
        {
            CCF2 = 0;                           //清PCA模块1中断标志
            if(P27)   CCAP2_tmp += PWM2_high;
                                        //输出为高电平，则给影射寄存器装载高电平时间长度
            else      CCAP2_tmp += PWM2_low;
                                        //输出为低电平，则给影射寄存器装载低电平时间长度
            CCAP2L = (u8)CCAP2_tmp;
                                        //将影射寄存器写入捕获寄存器，先写CCAP0L
            CCAP2H = (u8)(CCAP2_tmp >> 8);          //后写CCAP0H
        }
}
```

STC MCU Limited

2、汇编语言程序

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------*/

;**************   功能说明          **************

; 输出3路9~16位PWM信号。

; PWM频率 = MAIN_Fosc / PWM_DUTY, 假设 MAIN_Fosc = 24MHZ, PWM_DUTY = 6000, 则输出PWM频率为4000HZ.

; ****************************************

;*****************************用户宏定义*************************************************
Fosc_KHZ        EQU     24000           //定义主时钟, KHZ

PWM_DUTY        EQU     6000            //定义PWM的周期，数值为PCA所选择的时钟脉冲个数。
PWM_HIGH_MIN            EQU     80
                        //限制PWM输出的最小占空比, 避免中断里重装参数时间不够。
PWM_HIGH_MAX            EQU     (PWM_DUTY - PWM_HIGH_MIN)
                        //限制PWM输出的最大占空比。

;*********************************************************************************************

PCA0                            EQU     0
PCA1                            EQU     1
PCA2                            EQU     2
PCA_Counter                     EQU     3
PCA_P12_P11_P10_P37             EQU     (0 SHL 4)
PCA_P34_P35_P36_P37             EQU     (1 SHL 4)
PCA_P24_P25_P26_P27             EQU     (2 SHL 4)
PCA_Mode_Capture                EQU     0
PCA_Mode_SoftTimer              EQU     048H
PCA_Mode_HighPulseOutput        EQU     04CH
PCA_Clock_1T                    EQU     (4 SHL 1)
PCA_Clock_2T                    EQU     (1 SHL 1)
PCA_Clock_4T                    EQU     (5 SHL 1)
PCA_Clock_6T                    EQU     (6 SHL 1)
PCA_Clock_8T                    EQU     (7 SHL 1)
PCA_Clock_12T                   EQU     (0 SHL 1)
PCA_Clock_ECI                   EQU     (3 SHL 1)
PCA_Rise_Active                 EQU     (1 SHL 5)
PCA_Fall_Active                 EQU     (1 SHL 4)
```

STC MCU Limited

```
ENABLE          EQU     1

AUXR1           DATA    0xA2
CCON            DATA    0xD8
CMOD            DATA    0xD9
CCAPM0          DATA    0xDA            ; PCA模块0的工作模式寄存器。
CCAPM1          DATA    0xDB            ; PCA模块1的工作模式寄存器。
CCAPM2          DATA    0xDC            ; PCA模块2的工作模式寄存器。

CL              DATA    0xE9
CCAP0L          DATA    0xEA            ; PCA模块0的捕捉/比较寄存器低8位。
CCAP1L          DATA    0xEB            ; PCA模块1的捕捉/比较寄存器低8位。
CCAP2L          DATA    0xEC            ; PCA模块2的捕捉/比较寄存器低8位。

CH              DATA    0xF9
CCAP0H          DATA    0xFA            ; PCA模块0的捕捉/比较寄存器高8位。
CCAP1H          DATA    0xFB            ; PCA模块1的捕捉/比较寄存器高8位。
CCAP2H          DATA    0xFC            ; PCA模块2的捕捉/比较寄存器高8位。

CCF0            BIT     CCON.0          ; PCA 模块0中断标志，由硬件置位，必须由软件清0。
CCF1            BIT     CCON.1          ; PCA 模块1中断标志，由硬件置位，必须由软件清0。
CCF2            BIT     CCON.2          ; PCA 模块2中断标志，由硬件置位，必须由软件清0。
CR              BIT     CCON.6          ; 1: 允许PCA计数器计数，0: 禁止计数。
CF              BIT     CCON.7
                                        ; PCA计数器溢出（CH，CL由FFFFH变为0000H）标志。
                                        ;PCA计数器溢出后由硬件置位，必须由软件清0。
PPCA            BIT     IP.7            ; PCA 中断 优先级设定位

P2M1            DATA    095H            ; P2M1.n,P2M0.n   =00--->Standard,    01--->push-pull
P2M0            DATA    096H            ;                 =10--->pure input,  11--->open drain

;=========================================================================

P25 BIT P2.5
P26 BIT P2.6
P27 BIT P2.7

PWM0_high_H             DATA    030H
PWM0_high_L             DATA    031H
PWM0_low_H              DATA    032H
PWM0_low_L              DATA    033H

PWM1_high_H             DATA    034H
PWM1_high_L             DATA    035H
PWM1_low_H              DATA    036H
PWM1_low_L              DATA    037H

PWM2_high_H             DATA    038H
```

STC MCU Limited

```
PWM2_high_L             DATA    039H
PWM2_low_H              DATA    03AH
PWM2_low_L              DATA    03BH

pwm0_H                  DATA    03CH
pwm0_L                  DATA    03DH
pwm1_H                  DATA    03EH
pwm1_L                  DATA    03FH
pwm2_H                  DATA    040H
pwm2_L                  DATA    041H

STACK_POIRTER           EQU     0D0H            ;堆栈开始地址

;*******************************************************************
;*******************************************************************
        ORG     00H                             ;reset
        LJMP    F_Main

        ORG     3BH                             ;7  PCA interrupt
        LJMP    F_PCA_Interrupt

;******************** 主程序 **************************/
F_Main:

        MOV     SP,     #STACK_POIRTER
        MOV     PSW,    #0
        USING   0                               ;选择第0组R0~R7

;================= 用户初始化程序 ====================================
        LCALL   F_PCA_Init                      ;PCA初始化
        SETB    EA
        ANL     P2M1,   #NOT 0E0H               ; P2.7 P2.6 P2.5 设置为推挽输出
        ORL     P2M0,   #0E0H

;================== 主循环 ===================================
L_MainLoop:
        MOV             R7, #2
        LCALL           F_delay_ms

        MOV             A, pwm0_L
                        ; if(++pwm0 >= PWM_HIGH_MAX)   pwm0 = PWM_HIGH_MIN
        ADD             A, #1
        MOV             pwm0_L, A
        MOV             A, pwm0_H
        ADDC            A, #0
        MOV             pwm0_H, A
        MOV             A, pwm0_L
        CLR     C
        SUBB    A,      #LOW PWM_HIGH_MAX
```

STC MCU Limited

```
        MOV     A,          pwm0_H
        SUBB    A,          #HIGH PWM_HIGH_MAX
        JC      L_PWM0_NotOverFollow
        MOV     pwm0_H, #HIGH PWM_HIGH_MIN
        MOV     pwm0_L, #LOW PWM_HIGH_MIN
L_PWM0_NotOverFollow:
        MOV     R5, #PCA0
        MOV     R6, pwm0_H
        MOV     R7, pwm0_L
        LCALL   F_PWMn_Update

        MOV     A, pwm1_L
                                ; if(++pwm1 >= PWM_HIGH_MAX)   pwm1 = PWM_HIGH_MIN
        ADD     A, #1
        MOV     pwm1_L, A
        MOV     A, pwm1_H
        ADDC    A, #0
        MOV     pwm1_H, A
        MOV     A, pwm1_L
        CLR     C
        SUBB    A, #LOW PWM_HIGH_MAX
        MOV     A, pwm1_H
        SUBB    A, #HIGH PWM_HIGH_MAX
        JC      L_PWM1_NotOverFollow
        MOV     pwm1_H, #HIGH PWM_HIGH_MIN
        MOV     pwm1_L, #LOW PWM_HIGH_MIN
L_PWM1_NotOverFollow:
        MOV     R5, #PCA1
        MOV     R6, pwm1_H
        MOV     R7, pwm1_L
        LCALL   F_PWMn_Update

        MOV     A, pwm2_L
                                ; if(++pwm2 >= PWM_HIGH_MAX)   pwm2 = PWM_HIGH_MIN
        ADD     A, #1
        MOV     pwm2_L, A
        MOV     A, pwm2_H
        ADDC    A, #0
        MOV     pwm2_H, A
        MOV     A, pwm2_L
        CLR     C
        SUBB    A, #LOW PWM_HIGH_MAX
        MOV     A, pwm2_H
        SUBB    A, #HIGH PWM_HIGH_MAX
        JC      L_PWM2_NotOverFollow
        MOV     pwm2_H, #HIGH PWM_HIGH_MIN
        MOV     pwm2_L, #LOW PWM_HIGH_MIN
L_PWM2_NotOverFollow:
        MOV     R5, #PCA2
```

STC MCU Limited

```
        MOV     R6, pwm2_H
        MOV     R7, pwm2_L
        LCALL   F_PWMn_Update

        LJMP    L_MainLoop

;//========================================================================
;// 函数: F_delay_ms
;// 描述: 延时子程序。
;// 参数: R7: 延时ms数.
;// 返回: none.
;// 版本: VER1.0
;// 日期: 2013-4-1
;// 备注: 除了ACCC和PSW外, 所用到的通用寄存器都入栈
;//========================================================================
F_delay_ms:
        PUSH    AR3             ;入栈R3
        PUSH    AR4             ;入栈R4

L_delay_ms_1:
        MOV     R3, #HIGH (Fosc_KHZ / 13)
        MOV     R4, #LOW (Fosc_KHZ / 13)

L_delay_ms_2:
        MOV     A, R4           ;1T             Total 13T/loop
        DEC     R4              ;2T
        JNZ     L_delay_ms_3    ;4T
        DEC     R3
L_delay_ms_3:
        DEC     A               ;1T
        ORL     A, R3           ;1T
        JNZ     L_delay_ms_2    ;4T

        DJNZ    R7, L_delay_ms_1

        POP     AR4             ;出栈R2
        POP     AR3             ;出栈R3
        RET

;========================================================================
; 函数: F_PWMn_Update
; 描述: 更新占空比数据。
; 参数: R5: PCA通道数。
;       R6,R7: PWM值.
; 返回: 无
; 版本: VER1.0
; 日期: 2014-2-15
; 备注:
;========================================================================
```

STC MCU Limited

```
F_PWMn_Update:
        PUSH    AR3
        PUSH    AR4

        CLR     C
        MOV     A, R7
        SUBB    A, #LOW PWM_HIGH_MAX
        MOV     A, R6
        SUBB    A, #HIGH PWM_HIGH_MAX
        JC      L_QuitCheckPwm_1
        MOV     R6, #HIGH PWM_HIGH_MAX
                                        ; 如果写入大于最大占空比数据，强制为最大占空比。
        MOV     R7, #LOW  PWM_HIGH_MAX
L_QuitCheckPwm_1:

        CLR     C
        MOV     A, R7
        SUBB    A, #LOW PWM_HIGH_MIN
        MOV     A, R6
        SUBB    A, #HIGH PWM_HIGH_MIN
        JNC     L_QuitCheckPwm_2
        MOV     R6, #HIGH PWM_HIGH_MIN
                                        ; 如果写入小于最小占空比数据，强制为最小占空比。
        MOV     R7, #LOW  PWM_HIGH_MIN
L_QuitCheckPwm_2:

        CLR     C
        MOV     A, #LOW PWM_DUTY        ;计算并保存PWM输出低电平的PCA时钟脉冲个数
        SUBB    A, R7
        MOV     R4, A
        MOV     A, #HIGH PWM_DUTY
        SUBB    A, R6
        MOV     R3, A

        CJNE    R5, #PCA0, L_NotLoadPCA0
        CLR     CR                      ; 停止PCA一会， 一般不会影响PWM。
        MOV     PWM0_high_H, R6         ; 数据装入占空比变量。
        MOV     PWM0_high_L, R7
        MOV     PWM0_low_H, R3
        MOV     PWM0_low_L, R4
        SETB    CR                      ; 启动PCA。
L_NotLoadPCA0:

        CJNE    R5, #PCA1, L_NotLoadPCA1
        CLR     CR                      ; 停止PCA一会， 一般不会影响PWM。
        MOV     PWM1_high_H, R6                 ; 数据装入占空比变量。
        MOV     PWM1_high_L, R7
        MOV     PWM1_low_H, R3
        MOV     PWM1_low_L, R4
```

STC MCU Limited

```
        SETB    CR                              ; 启动PCA。
L_NotLoadPCA1:

        CJNE    R5, #PCA2, L_NotLoadPCA2
        CLR     CR                              ; 停止PCA一会， 一般不会影响PWM。
        MOV     PWM2_high_H, R6                 ; 数据装入占空比变量。
        MOV     PWM2_high_L, R7
        MOV     PWM2_low_H, R3
        MOV     PWM2_low_L, R4
        SETB    CR                              ; 启动PCA。
L_NotLoadPCA2:

        POP     AR4
        POP     AR3
        RET

;==========================================================================
; 函数: F_PCA_Init
; 描述: PCA初始化程序.
; 参数: none
; 返回: none.
; 版本: V1.0, 2013-11-22
;==========================================================================
F_PCA_Init:
        CLR     CR
        MOV     CH, #0
        MOV     CL, #0
        MOV     A, AUXR1
        ANL     A, # NOT(3 SHL 4)
        ORL     A, #PCA_P24_P25_P26_P27         ;切换IO口
        MOV     AUXR1, A
        ANL     A, #NOT(7 SHL 1)
        ORL     A, #PCA_Clock_1T                ;选择时钟源
        MOV     CMOD, A

        MOV     CCAPM0, #(PCA_Mode_HighPulseOutput OR ENABLE)
                                                ; 16位软件定时、高速脉冲输出、中断模式
        MOV     CCAPM1, #(PCA_Mode_HighPulseOutput OR ENABLE)
                                                ; 16位软件定时、高速脉冲输出、中断模式
        MOV     CCAPM2, #(PCA_Mode_HighPulseOutput OR ENABLE)
                                                ; 16位软件定时、高速脉冲输出、中断模式

        MOV     pwm0_H, #HIGH (PWM_DUTY / 4 * 1)        ;给PWM一个初值
        MOV     pwm0_L, #LOW  (PWM_DUTY / 4 * 1)
        MOV     pwm1_H, #HIGH (PWM_DUTY / 4 * 2)
        MOV     pwm1_L, #LOW  (PWM_DUTY / 4 * 2)
        MOV     pwm2_H, #HIGH (PWM_DUTY / 4 * 3)
        MOV     pwm2_L, #LOW  (PWM_DUTY / 4 * 3)
```

STC MCU Limited

```
        MOV     R5, #PCA0
        MOV     R6, pwm0_H
        MOV     R7, pwm0_L
        LCALL   F_PWMn_Update
        MOV     R5, #PCA1
        MOV     R6, pwm1_H
        MOV     R7, pwm1_L
        LCALL   F_PWMn_Update
        MOV     R5, #PCA2
        MOV     R6, pwm2_H
        MOV     R7, pwm2_L
        LCALL   F_PWMn_Update

        SETB    PPCA                    ; 高优先级中断
        SETB    CR                      ; 运行PCA定时器
        RET

;========================================================================
;========================================================================
; 函数: F_PCA_Interrupt
; 描述: PCA中断处理程序.
; 参数: None
; 返回: none.
; 版本: V1.0, 2012-11-22
;========================================================================
F_PCA_Interrupt:
        PUSH    PSW
        PUSH    ACC

        ;================ PCA模块0中断 ====================
        JNB     CCF0, L_QuitPCA0        ; PCA模块0中断
        CLR     CCF0                    ; 清PCA模块0中断标志

        JNB     P25, L_PCA0_LoadLow
        MOV     A, CCAP0L               ; 输出为高电平，则给影射寄存器装载高电平时间长度
        ADD     A, PWM0_high_L          ; 加上高电平时间，
        MOV     CCAP0L, A               ; 先写CCAP0L
        MOV     A, CCAP0H               ;
        ADDC    A, PWM0_high_H ;
        MOV     CCAP0H, A               ; 后写CCAP0H
        SJMP    L_QuitPCA0
L_PCA0_LoadLow:
        MOV     A, CCAP0L               ; 输出为低电平，则给影射寄存器装载低电平时间长度
        ADD     A, PWM0_low_L           ; 加上低电平时间，
        MOV     CCAP0L, A                       ; 先写CCAP0L
        MOV     A, CCAP0H                       ;
        ADDC    A, PWM0_low_H                   ;
        MOV     CCAP0H, A                       ; 后写CCAP0H
L_QuitPCA0:
```

STC MCU Limited

```
        ;=============== PCA模块1中断 ====================
        JNB     CCF1, L_QuitPCA1        ; PCA模块1中断
        CLR     CCF1                    ; 清PCA模块1中断标志

        JNB     P26, L_PCA1_LoadLow
        MOV     A, CCAP1L               ; 输出为高电平，则给影射寄存器装载高电平时间长度
        ADD     A, PWM1_high_L          ; 加上高电平时间，
        MOV     CCAP1L, A               ; 先写CCAP1L
        MOV     A, CCAP1H               ;
        ADDC    A, PWM1_high_H ;
        MOV     CCAP1H, A               ; 后写CCAP1H
        SJMP    L_QuitPCA1
L_PCA1_LoadLow:
        MOV     A, CCAP1L               ; 输出为低电平，则给影射寄存器装载低电平时间长度
        ADD     A, PWM1_low_L           ; 加上低电平时间，
        MOV     CCAP1L, A               ; 先写CCAP1L
        MOV     A, CCAP1H               ;
        ADDC    A, PWM1_low_H  ;
        MOV     CCAP1H, A               ; 后写CCAP1H
L_QuitPCA1:

        ;=============== PCA模块2中断 ====================
        JNB     CCF2, L_QuitPCA2        ; PCA模块2中断
        CLR     CCF2                    ; 清PCA模块2中断标志

        JNB     P27, L_PCA2_LoadLow
        MOV     A, CCAP2L               ; 输出为高电平，则给影射寄存器装载高电平时间长度
        ADD     A, PWM2_high_L          ; 加上高电平时间，
        MOV     CCAP2L, A               ; 先写CCAP2L
        MOV     A, CCAP2H               ;
        ADDC    A, PWM2_high_H ;
        MOV     CCAP2H, A               ; 后写CCAP2H
        SJMP    L_QuitPCA2
L_PCA2_LoadLow:
        MOV     A, CCAP2L               ; 输出为低电平，则给影射寄存器装载低电平时间长度
        ADD     A, PWM2_low_L           ; 加上低电平时间，
        MOV     CCAP2L, A               ; 先写CCAP2L
        MOV     A, CCAP2H               ;
        ADDC    A, PWM2_low_H  ;
        MOV     CCAP2H, A               ; 后写CCAP2H
L_QuitPCA2:

        POP     ACC
        POP             PSW

        RETI
        END
```

STC MCU Limited

11.9 用CCP/PCA的16位捕获模式测脉冲宽度的程序(C和汇编)

1. C程序:

```
/*------------------------------------------------------------------------------------------------------ */
/* --- STC MCU  Limited. ------------------------------------------------------------------------------ */
/* --- STC15F2K60S2 系列 PCA实现16位捕获举例------------------------------------------------------------ */
/* --- 研发顾问QQ：800003751-------------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ----------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                  */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                  */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*------------------------------------------------------------------------------------------------------ */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC    18432000L

typedef unsigned char      BYTE;
typedef unsigned int       WORD;
typedef unsigned long      DWORD;


sfr     P_SW1   =       0xA2;                   //外设功能切换寄存器1

#define CCP_S0          0x10                    //P_SW1.4
#define CCP_S1          0x20                    //P_SW1.5

sfr     CCON    =       0xD8;                   //PCA控制寄存器
sbit    CCF0    =       CCON^0;                 //PCA模块0中断标志
sbit    CCF1    =       CCON^1;                 //PCA模块1中断标志
sbit    CR      =       CCON^6;                 //PCA定时器运行控制位
sbit    CF      =       CCON^7;                 //PCA定时器溢出标志
sfr     CMOD    =       0xD9;                   //PCA模式寄存器
sfr     CL      =       0xE9;                   //PCA定时器低字节
sfr     CH      =       0xF9;                   //PCA定时器高字节
sfr     CCAPM0          =       0xDA;           //PCA模块0模式寄存器
sfr     CCAP0L          =       0xEA;           //PCA模块0捕获寄存器 LOW
sfr     CCAP0H          =       0xFA;           //PCA模块0捕获寄存器 HIGH
sfr     CCAPM1          =       0xDB;           //PCA模块1模式寄存器
sfr     CCAP1L          =       0xEB;           //PCA模块1捕获寄存器 LOW
sfr     CCAP1H          =       0xFB;           //PCA模块1捕获寄存器 HIGH
sfr     CCAPM2          =       0xDC;           //PCA模块2模式寄存器
sfr     CCAP2L          =       0xEC;           //PCA模块2捕获寄存器 LOW
```

STC MCU Limited

```
sfr     CCAP2H      =  0xFC;                //PCA模块2捕获寄存器 HIGH
sfr     PCA_PWM0    =  0xf2;                //PCA模块0的PWM寄存器
sfr     PCA_PWM1    =  0xf3;                //PCA模块1的PWM寄存器
sfr     PCA_PWM2    =  0xf4;                //PCA模块2的PWM寄存器

BYTE    cnt;                                //存储PCA计时溢出次数
DWORD count0;                               //记录上一次的捕获值
DWORD count1;                               //记录本次的捕获值
DWORD length;                               //存储信号的时间长度(count1 - count0)

void main()
{
        ACC     =       P_SW1;
        ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=0
        P_SW1   =       ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      ACC     =       P_SW1;
//      ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=1 CCP_S1=0
//      ACC     |=      CCP_S0;         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      P_SW1   =       ACC;
//
//      ACC     =       P_SW1;
//      ACC     &=      ~(CCP_S0 | CCP_S1);     //CCP_S0=0 CCP_S1=1
//      ACC     |=      CCP_S1;         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      P_SW1   =       ACC;

        CCON    =       0;                      //初始化PCA控制寄存器
                                                //PCA定时器停止
                                                //清除CF标志
                                                //清除模块中断标志
        CL      =       0;                      //复位PCA寄存器
        CH      =       0;
        CCAP0L  =       0;
        CCAP0H  =       0;
        CMOD    =       0x09;           //设置PCA时钟源为系统时钟,且使能PCA计时溢出中断
        CCAPM0  =       0x21;           //PCA模块0为16位捕获模式(上升沿捕获,
                                        //可测从高电平开始的整个周期),且产生捕获中断
//      CCAPM0  =       0x11;           //PCA模块0为16位捕获模式(下降沿捕获,
                                        //可测从低电平开始的整个周期),且产生捕获中断
//      CCAPM0  =       0x31;           //PCA模块0为16位捕获模式(上升沿/下降沿捕获,
                                        //可测高电平或者低电平宽度),且产生捕获中断

        CR      =       1;              //PCA定时器开始工作

        EA      =       1;

        cnt     =       0;
        count0  =       0;
        count1  =       0;
```

STC MCU Limited

```
        while (1);
}

void PCA_isr() interrupt 7 using 1
{
        if (CF)
        {
                CF = 0;
                cnt++;                                  //PCA计时溢出次数+1
        }
        if (CCF0)
        {
                CCF0 = 0;
                count0 = count1;                        //备份上一次的捕获值
                ((BYTE *)&count1)[3] = CCAP0L;          //保存本次的捕获值
                ((BYTE *)&count1)[2] = CCAP0H;
                ((BYTE *)&count1)[1] = cnt;
                ((BYTE *)&count1)[0] = 0;
                length = count1 - count0;               //计算两次捕获的差值,即得到时间长度
        }
}
```

STC MCU Limited

2. 汇编程序：

```
/*---------------------------------------------------------------------------------------------------- */
/* --- STC MCU  Limited. ------------------------------------------------------------------------------ */
/* --- STC15F2K60S2 系列 PCA实现16位捕获举例---------------------------------------------------------- */
/* --- 研发顾问QQ：800003751--------------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 --------------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 --------------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ---------------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序----                                    */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序----                                    */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

P_SW1           EQU     0A2H            ;外设功能切换寄存器1

CCP_S0          EQU     10H             ;P_SW1.4
CCP_S1          EQU     20H             ;P_SW1.5

CCON            EQU     0D8H            ;PCA控制寄存器
CCF0            BIT     CCON.0          ;PCA模块0中断标志
CCF1            BIT     CCON.1          ;PCA模块1中断标志
CR              BIT     CCON.6          ;PCA定时器运行控制位
CF              BIT     CCON.7          ;PCA定时器溢出标志
CMOD            EQU     0D9H            ;PCA模式寄存器
CL              EQU     0E9H            ;PCA定时器低字节
CH              EQU     0F9H            ;PCA定时器高字节
CCAPM0          EQU     0DAH            ;PCA模块0模式寄存器
CCAP0L          EQU     0EAH            ;PCA模块0捕获寄存器 LOW
CCAP0H          EQU     0FAH            ;PCA模块0捕获寄存器 HIGH
CCAPM1          EQU     0DBH            ;PCA模块1模式寄存器
CCAP1L          EQU     0EBH            ;PCA模块1捕获寄存器 LOW
CCAP1H          EQU     0FBH            ;PCA模块1捕获寄存器 HIGH
CCAPM2          EQU     0DCH            ;PCA模块2模式寄存器
CCAP2L          EQU     0ECH            ;PCA模块2捕获寄存器 LOW
CCAP2H          EQU     0FCH            ;PCA模块2捕获寄存器 HIGH
PCA_PWM0        EQU     0F2H            ;PCA模块0的PWM寄存器
PCA_PWM1        EQU     0F3H            ;PCA模块1的PWM寄存器
PCA_PWM2        EQU     0F4H            ;PCA模块2的PWM寄存器


CNT             EQU     30H             ;存储PCA计时溢出次数
COUNT0          EQU     31H             ;记录上一次的捕获值,3字节
COUNT1          EQU     34H             ;记录本次的捕获值,3字节
LENGTH          EQU     37H             ;存储信号的时间长度,3字节(count1 - count0)

;----------------------------------------
```

STC MCU Limited

```
        ORG     0000H
        LJMP    MAIN
        ORG     003BH
PCA_ISR:
        PUSH    PSW
        PUSH    ACC
        JNB     CF,     CKECK_CCF0              ;判断是否为PCA计时溢出中断
        CLR     CF
        INC     CNT                             ;PCA计时溢出次数+1
CKECK_CCF0:
        JNB     CCF0,   PCA_ISR_EXIT            ;判断是否为捕获中断
        CLR     CCF0
        MOV     COUNT0,         COUNT1          ;备份上一次的捕获值
        MOV     COUNT0+1,       COUNT1+1
        MOV     COUNT0+2,       COUNT1+2
        MOV     COUNT1,         CNT             ;保存本次的捕获值
        MOV     COUNT1+1,       CCAP0H
        MOV     COUNT1+2,       CCAP0L
        CLR     C                               ;计算两次捕获的差值
        MOV     A,      COUNT1+2
        SUBB    A,      COUNT0+2
        MOV     LENGTH+2,       A
        MOV     A,      COUNT1+1
        SUBB    A,      COUNT0+1
        MOV     LENGTH+1,       A
        MOV     A,      COUNT1
        SUBB    A,      COUNT0
        MOV     LENGTH,         A               ;LENGTH存放的即为时间长度
PCA_ISR_EXIT:
        POP     ACC
        POP     PSW
        RETI

;----------------------------------------

        ORG     0100H
MAIN:
        MOV     SP,     #5FH

        MOV     A,      P_SW1
        ANL     A,      #0CFH                   //CCP_S0=0 CCP_S1=0
        MOV     P_SW1,  A                       //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//      MOV     A,      P_SW1
//      ANL     A,      #0CFH                   //CCP_S0=1 CCP_S1=0
//      ORL     A,      #CCP_S0         //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//      MOV     P_SW1,  A
//
//      MOV     A,      P_SW1
```

STC MCU Limited

```
//      ANL     A,      #0CFH                   //CCP_S0=0 CCP_S1=1
//      ORL     A,      #CCP_S1         //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//      MOV     P_SW1,  A

        MOV     CCON,   #0                      ;初始化PCA控制寄存器
                                                ;PCA定时器停止
                                                ;清除CF标志
                                                ;清除模块中断标志
        CLR     A                               ;
        MOV     CL,     A                       ;复位PCA计时器
        MOV     CH,     A                       ;
        MOV     CCAP0L,         A
        MOV     CCAP0H,         A
        MOV     CMOD,   #09H                    ;设置PCA时钟源为系统时钟
                                                ;使能PCA定时器溢出中断
        MOV     CCAPM0,         #21H            ;PCA模块0为16位捕获模式(上升沿捕获,
                                                ;可测从高电平开始的整个周期),且产生捕获中断
        MOV     CCAPM0,         #11H            ;PCA模块0为16位捕获模式(下降沿捕获,
                                                ;可测从低电平开始的整个周期),且产生捕获中断
        MOV     CCAPM0,         #31H            ;PCA模块0为16位捕获模式(上升沿/下降沿捕获,
                                                ;可测高电平或者低电平宽度),且产生捕获中断

        SETB    CR                              ;PCA定时器开始工作

        SETB    EA

        CLR     A                               ;初始化变量
        MOV     CNT,    A
        MOV     COUNT0,         A
        MOV     COUNT0+1,       A
        MOV     COUNT0+2,       A
        MOV     COUNT1,         A
        MOV     COUNT1+1,       A
        MOV     COUNT1+2,       A
        MOV     LENGTH,         A
        MOV     LENGTH+1,       A
        MOV     LENGTH+2,       A

        SJMP    $
;---------------------------------------
        END
```

STC MCU Limited

11.10 用T0软硬结合模拟16路软件PWM的程序(C及汇编)

1. C程序：

```
/*-----------------------------------------------------------------------------------------------*/
/* --- STC MCU International Limited ----------------------------------------------------*/
/* --- STC 1T Series MCU Demo Programme ----------------------------------------------*/
/* --- Mobile: (86)13922805190 ----------------------------------------------------------*/
/* --- Fax: 86-0513-55012956,55012947,55012969 ----------------------------------------*/
/* --- Tel: 86-0513-55012928,55012929,55012966 ----------------------------------------*/
/* --- Web: www.GXWMCU.com ---------------------------------------------------------*/
/* --- QQ:  800003751 ------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/*-----------------------------------------------------------------------------------------------*/

/*************    功能说明          **************

使用Timer0模拟16通道PWM驱动程序。

输出为 P1.0 ~ P1.7, P2.0 ~ P2.7, 对应 PWM0 ~ PWM15.

定时器中断频率一般不要超过100KHZ, 留足够的时间给别的程序运行.

本例子使用22.1184MHZ时钟, 25K的中断频率, 250级PWM, 周期为10ms.

中断里处理的时间不超过6us, 占CPU时间大约为15%.

*******************************************/

#include  <reg52.h>

#define MAIN_Fosc          22118400UL                                  //定义主时钟
#define   Timer0_Rate       25000                                       //中断频率

typedef   unsigned char     u8;
typedef   unsigned int      u16;
typedef   unsigned long     u32;

sfr       AUXR = 0x8E;

#define   Timer0_Reload     (65536UL -(MAIN_Fosc / Timer0_Rate))        //Timer 0 重装值

//************** PWM8 变量和常量以及IO口定义 ***************
//*******************    8通道8 bit 软PWM*******************

#define   PWM_DUTY_MAX          250          // 0~255  PWM周期, 最大255
#define   PWM_ON                1            // 定义占空比的电平, 1 或 0

#define   PWM_OFF           (!PWM_ON)
```

STC MCU Limited

```
#define   PWM_ALL_ON    (0xff * PWM_ON)

u8 bdata  PWM_temp1, PWM_temp2;                   //影射一个RAM，可位寻址，输出时同步刷新
sbit      P_PWM0          =       PWM_temp1^0;    //          定义影射RAM每位对应的IO
sbit      P_PWM1          =       PWM_temp1^1;
sbit      P_PWM2          =       PWM_temp1^2;
sbit      P_PWM3          =       PWM_temp1^3;
sbit      P_PWM4          =       PWM_temp1^4;
sbit      P_PWM5          =       PWM_temp1^5;
sbit      P_PWM6          =       PWM_temp1^6;
sbit      P_PWM7          =       PWM_temp1^7;
sbit      P_PWM8          =       PWM_temp2^0;
sbit      P_PWM9          =       PWM_temp2^1;
sbit      P_PWM10         =       PWM_temp2^2;
sbit      P_PWM11         =       PWM_temp2^3;
sbit      P_PWM12         =       PWM_temp2^4;
sbit      P_PWM13         =       PWM_temp2^5;
sbit      P_PWM14         =       PWM_temp2^6;
sbit      P_PWM15         =       PWM_temp2^7;

u8        pwm_duty;                               //周期计数值
u8        pwm[16];                                //pwm0~pwm15 为0至15路PWM的宽度值

bit       B_1ms;
u8        cnt_1ms;
u8        cnt_20ms;

/*********************************************/
void main(void)
{
        u8        i;

        AUXR |=  (1<<7);                          // Timer0 set as 1T mode
        TMOD &= ~(1<<2);                          // Timer0 set as Timer
        TMOD &= ~0x03;                            // Timer0 set as 16 bits Auto Reload
        TH0 = Timer0_Reload / 256;                //Timer0 Load
        TL0 = Timer0_Reload % 256;
        ET0 = 1;                                  //Timer0 Interrupt Enable
        PT0 = 1;                                  //高优先级
        TR0 = 1;                                  //Timer0 Run
        EA = 1;                                   //打开总中断

        cnt_1ms  = Timer0_Rate / 1000;            //1ms计数
        cnt_20ms = 20;

        for(i=0; i<16; i++)  pwm[i] = i * 15 + 15;        //给PWM一个初值

        while(1)
        {
```

STC MCU Limited

```
                if(B_1ms)                                   //1ms到
                {
                        B_1ms = 0;
                        if(--cnt_20ms == 0)                 //PWM 20ms改变一阶
                        {
                                cnt_20ms = 20;
                                for(i=0; i<16; i++)  pwm[i]++;
                        }
                }
        }
}
/********************** Timer0 1ms中断函数 ************************/
void timer0 (void) interrupt 1
{
        P1 = PWM_temp1;                                     //影射RAM输出到实际的PWM端口
        P2 = PWM_temp2;

        if(++pwm_duty == PWM_DUTY_MAX)                      //PWM周期结束，重新开始新的周期
        {
                pwm_duty = 0;
                PWM_temp1 = PWM_ALL_ON;
                PWM_temp2 = PWM_ALL_ON;
        }
        ACC = pwm_duty;
        if(ACC == pwm[0])       P_PWM0  = PWM_OFF;          //判断PWM占空比是否结束
        if(ACC == pwm[1])       P_PWM1  = PWM_OFF;
        if(ACC == pwm[2])       P_PWM2  = PWM_OFF;
        if(ACC == pwm[3])       P_PWM3  = PWM_OFF;
        if(ACC == pwm[4])       P_PWM4  = PWM_OFF;
        if(ACC == pwm[5])       P_PWM5  = PWM_OFF;
        if(ACC == pwm[6])       P_PWM6  = PWM_OFF;
        if(ACC == pwm[7])       P_PWM7  = PWM_OFF;
        if(ACC == pwm[8])       P_PWM8  = PWM_OFF;
        if(ACC == pwm[9])       P_PWM9  = PWM_OFF;
        if(ACC == pwm[10])      P_PWM10 = PWM_OFF;
        if(ACC == pwm[11])      P_PWM11 = PWM_OFF;
        if(ACC == pwm[12])      P_PWM12 = PWM_OFF;
        if(ACC == pwm[13])      P_PWM13 = PWM_OFF;
        if(ACC == pwm[14])      P_PWM14 = PWM_OFF;
        if(ACC == pwm[15])      P_PWM15 = PWM_OFF;

        if(--cnt_1ms == 0)
        {
                cnt_1ms  = Timer0_Rate / 1000;
                B_1ms = 1;                                  // 1ms标志
        }
}
```

STC MCU Limited

2. 汇编程序:

```
;/*---------------------------------------------------------------------------------------------------*/
;/* --- STC MCU Limited ------------------------------------------------------------------------------*/
;/* --- Mobile: (86)13922805190 ----------------------------------------------------------------------*/
;/* --- Fax: 86-755-82905966 -------------------------------------------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------------------------------------------*/
;/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
;/*---------------------------------------------------------------------------------------------------*/

;**************    功能说明            ***************

;使用Timer0模拟16通道PWM驱动程序。

;输出为 P1.0 ~ P1.7, P2.0 ~ P2.7, 对应 PWM0 ~ PWM15.

;定时器中断频率一般不要超过100KHZ, 留足够的时间给别的程序运行.

;本例子使用22.1184MHZ时钟, 25K的中断频率, 250级PWM, 周期为10ms.

;中断里处理的时间不超过6us, 占CPU时间大约为15%.

;*****************************************

Fosc_KHZ        EQU     22118           ; 定义主时钟 KHZ
Timer0_Rate     EQU     25              ; 中断频率, KHZ
PWM_DUTY_MAX    EQU     250             ; 0~255   PWM周期, 最大255

AUXR            DATA    08EH
Timer0_Reload   EQU     (65536 - (Fosc_KHZ / Timer0_Rate))      ;Timer 0 重装值

;************** PWM8 变量和常量以及IO口定义 ***************
;*********************   8通道8 bit 软PWM**********************

PWM_temp1       DATA    20H             ; 影射一个RAM，可位寻址，输出时同步刷新
PWM_temp2       DATA    21H             ; 影射一个RAM，可位寻址，输出时同步刷新
P_PWM0          BIT     PWM_temp1.0     ; 定义影射RAM每位对应的IO
P_PWM1          BIT     PWM_temp1.1
P_PWM2          BIT     PWM_temp1.2
P_PWM3          BIT     PWM_temp1.3
P_PWM4          BIT     PWM_temp1.4
P_PWM5          BIT     PWM_temp1.5
P_PWM6          BIT     PWM_temp1.6
P_PWM7          BIT     PWM_temp1.7
P_PWM8          BIT     PWM_temp2.0
P_PWM9          BIT     PWM_temp2.1
P_PWM10         BIT     PWM_temp2.2
P_PWM11         BIT     PWM_temp2.3
P_PWM12         BIT     PWM_temp2.4
```

STC MCU Limited

```
P_PWM13         BIT     PWM_temp2.5
P_PWM14         BIT     PWM_temp2.6
P_PWM15         BIT     PWM_temp2.7

B_1ms           BIT     22H.0

cnt_1ms         DATA    30H
cnt_20ms        DATA    31H

pwm_duty        DATA    32H                     ; 周期计数值
pwm             EQU     40H                     ; 40H ~ 4FH 为0至15路PWM的宽度值

STACK_POIRTER EQU       0D0H                    ;堆栈开始地址

;*******************************************************************
;*******************************************************************
        ORG     00H                             ;reset
        LJMP    F_Main

        ORG     0BH                             ;1  Timer0 interrupt
        LJMP    F_Timer0_Interrupt

;********************* 主程序 ***************************/
F_Main:

        MOV     SP, #STACK_POIRTER
        MOV     PSW, #0
        USING   0                               ;选择第0组R0~R7

;================== 用户初始化程序 ====================================

        ORL     AUXR, #(1<<7)                   ; Timer0 set as 1T mode
        ANL     TMOD, #NOT (1 SHL 2)            ; Timer0 set as Timer
        ANL     TMOD, #NOT 0x03                 ; Timer0 set as 16 bits Auto Reload
        MOV     TH0, #HIGH Timer0_Reload        ; Timer0 Load
        MOV     TL0, #LOW Timer0_Reload         ;
        SETB    ET0                             ; Timer0 Interrupt Enable
        SETB    PT0                             ; 高优先级
        SETB    TR0                             ; Timer0 Run
        SETB    EA                              ; 打开总中断

        MOV     cnt_1ms, #(Timer0_Rate / 1000)  ; 1ms计数
        MOV     cnt_20ms, #20

        MOV     R0, #pwm
        MOV     R2, #15
L_DefaultPWM:
        MOV     A, R2
        MOV     @R0, A                          ;给PWM一个初值
```

STC MCU Limited

```
        ADD     A, #15
        MOV     R2, A
        INC     R0
        CJNE    R0, #(pwm+16), L_DefaultPWM

;==================== 主循环 ===================================
L_MainLoop:
        JNB     B_1ms, $                        ; 等待1ms到
        CLR     B_1ms                           ;
        DJNZ    cnt_20ms, L_MainLoop
        MOV     cnt_20ms, #20                   ; PWM 20ms改变一阶

        MOV     R0, #pwm
L_SetPWM_Loop:
        INC     @R0                             ; pwm + 1
        INC     R0
        CJNE    R0, #(pwm+16), L_SetPWM_Loop

        SJMP    L_MainLoop

;********************** Timer0 1ms中断函数 ************************/
F_Timer0_Interrupt:

        PUSH    PSW
        PUSH    ACC

        MOV     P1, PWM_temp1                   ; 影射RAM输出到实际的PWM端口
        MOV     P2, PWM_temp2                   ;

        INC     pwm_duty
        MOV     A, pwm_duty
        CJNE    A, #PWM_DUTY_MAX, L_CheckPwm
        MOV     pwm_duty, #0                    ; PWM周期结束，重新开始新的周期
        CLR     A
        MOV     PWM_temp1, #0FFH
        MOV     PWM_temp2, #0FFH

L_CheckPwm:
        CJNE    A, pwm+0, $+5                   ;判断PWM占空比是否结束
        CLR     P_PWM0
        CJNE    A, pwm+1, $+5
        CLR     P_PWM1
        CJNE    A, pwm+2, $+5
        CLR     P_PWM2
        CJNE    A, pwm+3, $+5
        CLR     P_PWM3
        CJNE    A, pwm+4, $+5
        CLR     P_PWM4
        CJNE    A, pwm+5, $+5
```

STC MCU Limited

```
        CLR     P_PWM5
        CJNE    A, pwm+6, $+5
        CLR     P_PWM6
        CJNE    A, pwm+7, $+5
        CLR     P_PWM7
        CJNE    A, pwm+8, $+5
        CLR     P_PWM8
        CJNE    A, pwm+9, $+5
        CLR     P_PWM9
        CJNE    A, pwm+10, $+5
        CLR     P_PWM10
        CJNE    A, pwm+11, $+5
        CLR     P_PWM11
        CJNE    A, pwm+12, $+5
        CLR     P_PWM12
        CJNE    A, pwm+13, $+5
        CLR     P_PWM13
        CJNE    A, pwm+14, $+5
        CLR     P_PWM14
        CJNE    A, pwm+15, $+5
        CLR     P_PWM15

        DJNZ    cnt_1ms, L_QuitCheck_1ms
        MOV     cnt_1ms, #Timer0_Rate
        SETB    B_1ms                   ; 1ms标志
L_QuitCheck_1ms:

        POP     ACC
        POP     PSW

        RETI

        END
```

STC MCU Limited

11.11 用T0的时钟输出功能实现8~16位PWM的程序(C及汇编)——占用系统时间小于0.4%

下文为利用T0的时钟输出功能实现8~16位PWM（占用系统时间小于0.4%）的测试程序（包括C语言程序和汇编程序）。

1、C语言程序

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------------*/
/* --- STC 1T Series MCU RC Demo -------------------------------------------*/
/* --- Mobile: (86)13922805190 ---------------------------------------------*/
/* --- Fax: 86-0513-55012956,55012947,55012969 -----------------------------*/
/* --- Tel: 86-0513-55012928,55012929,55012966 -----------------------------*/
/* --- Web: www.GXWMCU.com -------------------------------------------------*/
/* --- QQ:  800003751 ------------------------------------------------------*/
/* If you want to use the program or the program referenced in the   */
/* article, please specify in which data and procedures from STC     */
/*---------------------------------------------------------------------------*/

#include  <reg52.h>

/*************    功能说明    **************

本程序演示使用定时器做软件PWM。

定时器0做16位自动重装，中断，从T0CLKO高速输出PWM。

本例程是使用STC15F/L系列MCU的定时器T0做模拟PWM的例程。

PWM可以是任意的量程。但是由于软件重装需要一点时间，所以PWM占空比最小为32T/周期，最大为(周期-32T)/周期, T为时钟周期。

PWM频率为周期的倒数。假如周期为6000, 使用24MHZ的主频，则PWM频率为4000HZ。

*****************************************/

#define   MAIN_Fosc       24000000UL      //定义主时钟

#define   PWM_DUTY        6000            //定义PWM的周期，数值为时钟周期数，假如使用
                                          //24.576MHZ的主频，则PWM频率为6000HZ。

#define   PWM_HIGH_MIN 32                 //限制PWM输出的最小占空比。用户请勿修改。
#define   PWM_HIGH_MAX          (PWM_DUTY-PWM_HIGH_MIN)
                                          //限制PWM输出的最大占空比。用户请勿修改。

typedef   unsigned char   u8;
typedef   unsigned int    u16;
```

STC MCU Limited

```
typedef   unsigned long     u32;

sfr       P3M1      =     0xB1;          //P3M1.n,P3M0.n   =00--->Standard,    01--->push-pull
sfr       P3M0      =     0xB2;          //                =10--->pure input,  11--->open drain
sfr       AUXR      =     0x8E;
sfr       INT_CLKO  =     0x8F;

sbit      P_PWM     =     P3^5;          //定义PWM输出引脚。
//sbit    P_PWM     =     P1^4;          //定义PWM输出引脚。STC15W204S

u16       pwm;                         //定义PWM输出高电平的时间的变量。用户操作PWM的变量。

u16       PWM_high,PWM_low;            //中间变量，用户请勿修改。

void      delay_ms(unsigned char ms);
void      LoadPWM(u16 i);


/********************* 主函数**************************/
void main(void)
{
        P_PWM = 0;
        P3M1 &= ~(1 << 5);             //P3.5 设置为推挽输出
        P3M0 |=  (1 << 5);

//      P1M1 &= ~(1 << 4);             //P1.4 设置为推挽输出       STC15W204S
//      P1M0 |=  (1 << 4);

        TR0 = 0;                       //停止计数
        ET0 = 1;                       //允许中断
        PT0 = 1;                       //高优先级中断
        TMOD &= ~0x03;                 //工作模式,0: 16位自动重装
        AUXR |=  0x80;                 //1T
        TMOD &= ~0x04;                 //定时
        INT_CLKO |=  0x01;             //输出时钟

        TH0 = 0;
        TL0 = 0;
        TR0 = 1;                       //开始运行

        EA = 1;

        pwm = PWM_DUTY / 10;           //给PWM一个初值，这里为10%占空比
        LoadPWM(pwm);                  //计算PWM重装值

        while (1)
        {
                while(pwm < (PWM_HIGH_MAX-8))
                {
```

STC MCU Limited

```
                pwm += 8;                       //PWM逐渐加到最大
                LoadPWM(pwm);
                delay_ms(8);
            }
            while(pwm > (PWM_HIGH_MIN+8))
            {
                pwm -= 8;           //PWM逐渐减到最小
                LoadPWM(pwm);
                delay_ms(8);
            }
        }
}

//========================================================================
// 函数: void  delay_ms(unsigned char ms)
// 描述: 延时函数。
// 参数: ms,要延时的ms数, 这里只支持1~255ms. 自动适应主时钟.
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//========================================================================
void  delay_ms(unsigned char ms)
{
    unsigned int i;
        do{
            i = MAIN_Fosc / 13000;
                while(--i)          ;
    }while(--ms);
}

/****************** 计算PWM重装值函数 ********************/
void    LoadPWM(u16 i)
{
        u16     j;

        if(i > PWM_HIGH_MAX)    i = PWM_HIGH_MAX;
                                        //如果写入大于最大占空比数据，则强制为最大占空比。
        if(i < PWM_HIGH_MIN)    i = PWM_HIGH_MIN;
                                        //如果写入小于最小占空比数据，则强制为最小占空比。
        j = 65536UL - PWM_DUTY + i;     //计算PWM低电平时间
        i = 65536UL - i;                //计算PWM高电平时间
        EA = 0;
        PWM_high = i;                   //装载PWM高电平时间
        PWM_low  = j;                   //装载PWM低电平时间
        EA = 1;
}

/********************** Timer0中断函数************************/
```

STC MCU Limited

```
void timer0_int (void) interrupt 1
{
        if(P_PWM)
        {
                TH0 = (u8)(PWM_low >> 8);          //如果是输出高电平，则装载低电平时间。
                TL0 = (u8)PWM_low;
        }
        else
        {
                TH0 = (u8)(PWM_high >> 8);         //如果是输出低电平，则装载高电平时间。
                TL0 = (u8)PWM_high;
        }
}
```

2、汇编语言程序

```
;/*------------------------------------------------------------------------------------------*/
;/* --- STC MCU Limited ------------------------------------------------------------------*/
;/* --- Mobile: (86)13922805190 ----------------------------------------------------------*/
;/* --- Fax: 86-755-82905966 -------------------------------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------------------------------*/
;/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
;/*------------------------------------------------------------------------------------------*/

;/*************   功能说明          **************

;本程序演示使用定时器做软件PWM。

;定时器0做16位自动重装，中断，从T0CLKO高速输出PWM。

;本例程是使用STC15F/L系列MCU的定时器T0做模拟PWM的例程。

;PWM可以是任意的量程。但是由于软件重装需要一点时间，所以PWM占空比最小为32T/周期，最大为(周期-32T)/周期, T为时钟周期。

;PWM频率为周期的倒数。假如周期为6000, 使用24MHZ的主频，则PWM频率为4000HZ。

;*****************************************/

;***************************用户宏定义*****************************************************
**
Fosc_KHZ        EQU     24000   //定义主时钟, KHZ

PWM_DUTY        EQU     6000    //定义PWM的周期，数值为PCA所选择的时钟脉冲个数。
PWM_HIGH_MIN    EQU     32      //限制PWM输出的最小占空比, 避免中断里重装参数时间不够。
PWM_HIGH_MAXEQU         (PWM_DUTY - PWM_HIGH_MIN)
                                //限制PWM输出的最大占空比。
```

STC MCU Limited

```
;*******************************************************************************************

P3M1            DATA 0B1H       ; P3M1.n,P3M0.n   =00--->Standard,    01--->push-pull
P3M0            DATA 0B2H       ;                 =10--->pure input,  11--->open drain
AUXR            DATA 08EH       ;
INT_CLKO        DATA 08FH       ;

P_PWM  BIT      P3.5            ; 定义PWM输出引脚。
;P_PWM BIT      P1.4            ; 定义PWM输出引脚。STC15W204S

pwm_H           DATA    030H    ; 定义PWM输出高电平的时间的变量。用户操作PWM的变量。
pwm_L           DATA    031H
PWM_high_H      DATA    032H    ; 中间变量，用户请勿修改
PWM_high_L      DATA    033H
PWM_low_H       DATA    034H
PWM_low_L       DATA    035H

STACK_POIRTER EQU       0D0H    ;堆栈开始地址

;*******************************************************************
;*******************************************************************

        ORG     00H                             ;reset
        LJMP    F_Main

        ORG     0BH                             ;1  Timer0 interrupt
        LJMP    F_Timer0_Interrupt

;********************* 主程序 ***************************/
F_Main:
        MOV     SP, #STACK_POIRTER
        MOV     PSW, #0
        USING   0                               ;选择第0组R0~R7

;================= 用户初始化程序 ====================================
        CLR     P_PWM
        ANL     P3M1, #NOT (1 SHL 5)            ; P3.5 设置为推挽输出
        ORL     P3M0, #(1 SHL 5);

;       ANL     P1M1, #NOT (1 SHL 4)            ; P1.4 设置为推挽输出       STC15W204S
;       ORL     P1M0, #(1 SHL 4)                ;

        CLR     TR0                             ; 停止计数
        SETB    ET0                             ; 允许中断
        SETB    PT0                             ; 高优先级中断
        ANL     TMOD, #NOT 003H                 ; 工作模式,0: 16位自动重装
        ORL     AUXR, #080H                     ; 1T
        ANL     TMOD, #NOT 004H                 ; 定时
```

STC MCU Limited

```
        ORL     INT_CLKO, #001H                 ; 输出时钟

        MOV     TH0, #0
        MOV     TL0, #0
        SETB    TR0                             ; 开始运行

        SETB    EA

        MOV     pwm_H, #HIGH (PWM_DUTY / 10)    ;给PWM一个初值，这里为10%占空比
        MOV     pwm_L, #LOW  (PWM_DUTY / 10)
        MOV     R6, pwm_H
        MOV     R7, pwm_L
        LCALL   F_PWMn_Update                   ; 计算PWM重装值

;================== 主循环 ==================================
L_MainLoop1:
        MOV     R7, #2
        LCALL   F_delay_ms

        MOV     A, pwm_L                        ; if(++pwm >= PWM_HIGH_MAX)
        ADD     A, #1
        MOV     pwm_L, A
        MOV     A, pwm_H
        ADDC    A, #0
        MOV     pwm_H, A
        MOV     A, pwm_L
        CLR     C
        SUBB    A, #LOW PWM_HIGH_MAX
        MOV     A, pwm_H
        SUBB    A, #HIGH PWM_HIGH_MAX
        JC      L_PWM_NotUpOverFollow           ; PWM逐渐加到最大
        MOV     pwm_H, #HIGH PWM_HIGH_MAX
        MOV     pwm_L, #LOW PWM_HIGH_MAX
        SJMP    L_MainLoop2
L_PWM_NotUpOverFollow:
        MOV     R6, pwm_H
        MOV     R7, pwm_L
        LCALL   F_PWMn_Update
        SJMP    L_MainLoop1

L_MainLoop2:
        MOV     R7, #2
        LCALL   F_delay_ms

        MOV     A, pwm_L                        ; if(++pwm < PWM_HIGH_MIN)
        CLR     C
        SUBB    A, #1
        MOV     pwm_L, A
        MOV     A, pwm_H
```

STC MCU Limited

```
        SUBB    A, #0
        MOV     pwm_H, A
        MOV     A, pwm_L
        CLR     C
        SUBB    A, #LOW PWM_HIGH_MIN
        MOV     A, pwm_H
        SUBB    A, #HIGH PWM_HIGH_MIN
        JNC     L_PWM_NotDnOverFollow                   ; PWM逐渐减到最小
        MOV     pwm_H, #HIGH PWM_HIGH_MIN
        MOV     pwm_L, #LOW PWM_HIGH_MIN
        SJMP    L_MainLoop1
L_PWM_NotDnOverFollow:
        MOV     R6, pwm_H
        MOV     R7, pwm_L
        LCALL   F_PWMn_Update
        SJMP    L_MainLoop2

;//=========================================================================
;// 函数: F_delay_ms
;// 描述: 延时子程序。
;// 参数: R7: 延时ms数.
;// 返回: none.
;// 版本: VER1.0
;// 日期: 2013-4-1
;// 备注: 除了ACCC和PSW外, 所用到的通用寄存器都入栈
;//=========================================================================
F_delay_ms:
        PUSH    AR3                             ;入栈R3
        PUSH    AR4                             ;入栈R4

L_delay_ms_1:
        MOV     R3, #HIGH (Fosc_KHZ / 13)
        MOV     R4, #LOW (Fosc_KHZ / 13)

L_delay_ms_2:
        MOV     A, R4                           ;1T             Total 13T/loop
        DEC     R4                              ;2T
        JNZ     L_delay_ms_3                    ;4T
        DEC     R3
L_delay_ms_3:
        DEC     A                               ;1T
        ORL     A, R3                           ;1T
        JNZ     L_delay_ms_2                    ;4T

        DJNZ    R7, L_delay_ms_1

        POP     AR4                             ;出栈R2
        POP     AR3                             ;出栈R3
        RET
```

STC MCU Limited

```
;========================================================================
; 函数: F_PWMn_Update
; 描述: 更新占空比数据。
; 参数: R6,R7: PWM值。
; 返回: 无
; 版本: VER1.0
; 日期: 2014-2-15
; 备注:
;========================================================================
F_PWMn_Update:
        PUSH    AR3
        PUSH    AR4

        CLR     C
        MOV     A, R7
        SUBB    A, #LOW PWM_HIGH_MAX
        MOV     A, R6
        SUBB    A, #HIGH PWM_HIGH_MAX
        JC      L_QuitCheckPwm_1
        MOV     R6, #HIGH PWM_HIGH_MAX
                                        ; 如果写入大于最大占空比数据，强制为最大占空比。
        MOV     R7, #LOW  PWM_HIGH_MAX
L_QuitCheckPwm_1:

        CLR     C
        MOV     A, R7
        SUBB    A, #LOW PWM_HIGH_MIN
        MOV     A, R6
        SUBB    A, #HIGH PWM_HIGH_MIN
        JNC     L_QuitCheckPwm_2
        MOV     R6, #HIGH PWM_HIGH_MIN
                                        ; 如果写入小于最小占空比数据，强制为最小占空比。
        MOV     R7, #LOW  PWM_HIGH_MIN
L_QuitCheckPwm_2:

        CLR     C
        MOV     A, R7                   ;计算并保存PWM输出低电平的T0时钟脉冲个数
        SUBB    A, #LOW PWM_DUTY
        MOV     R4, A
        MOV     A, R6
        SUBB    A, #HIGH PWM_DUTY
        MOV     R3, A

        CLR     C
        MOV     A, #0                   ;计算并保存PWM输出高电平的T0时钟脉冲个数
        SUBB    A, R7
        MOV     R7, A
        MOV     A, #0
        SUBB    A, R6
```

STC MCU Limited

```
        MOV     R6, A

        CLR     EA                      ;禁止一会中断， 一般不会影响PWM。
        MOV     PWM_high_H, R6          ;数据装入占空比变量。
        MOV     PWM_high_L, R7
        MOV     PWM_low_H, R3
        MOV     PWM_low_L, R4
        SETB    EA

        POP     AR4
        POP     AR3
        RET


;********************** Timer0中断函数************************/
F_Timer0_Interrupt:
        PUSH    PSW
        PUSH    ACC
        JNB     P_PWM, L_T0_LoadLow
        MOV     TH0, PWM_low_H          ;如果是输出高电平，则装载低电平时间。
        MOV     TL0, PWM_low_L
        SJMP    L_QuitTimer0
L_T0_LoadLow:
        MOV     TH0, PWM_high_H         ;如果是输出低电平，则装载高电平时间。
        MOV     TL0, PWM_high_L

L_QuitTimer0:

        POP     ACC
        POP     PSW

        RETI

        END
```

STC MCU Limited

11.12 用T1的时钟输出功能实现8~16位PWM的程序(C及汇编)——占用系统时间小于0.4%

下文为利用T1的时钟输出功能实现8~16位PWM（占用系统时间小于0.4%）的测试程序（包括C语言程序和汇编程序）。

1、C语言程序

```
/*---------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------*/
/* --- STC 1T Series MCU RC Demo -------------------------------------*/
/* --- Mobile: (86)13922805190 ---------------------------------------*/
/* --- Fax: 86-0513-55012956,55012947,55012969 -----------------------*/
/* --- Tel: 86-0513-55012928,55012929,55012966 -----------------------*/
/* --- Web: www.GXWMCU.com -------------------------------------------*/
/* --- QQ:  800003751 ------------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------*/

#include  <reg52.h>

/*************  功能说明            ***************
本程序演示使用定时器做软件PWM。

定时器1做16位自动重装，中断，从T1CLKO高速输出PWM。

本例程是使用STC15F/L系列MCU的定时器T1做模拟PWM的例程。

PWM可以是任意的量程。但是由于软件重装需要一点时间，所以PWM占空比最小为32T/周期，最大为(周期-32T)/周期, T为时钟周期。

PWM频率为周期的倒数。假如周期为6000, 使用24MHZ的主频，则PWM频率为4000HZ。

*****************************************/
#define   MAIN_Fosc       24000000UL      //定义主时钟

#define   PWM_DUTY        6000            //定义PWM的周期，数值为时钟周期数，假如使用
                                          //24.576MHZ的主频，则PWM频率为6000HZ。

#define   PWM_HIGH_MIN 32                 //限制PWM输出的最小占空比。用户请勿修改。
#define   PWM_HIGH_MAX          (PWM_DUTY-PWM_HIGH_MIN)
                                          //限制PWM输出的最大占空比。用户请勿修改。

typedef   unsigned char   u8;
typedef   unsigned int    u16;
typedef   unsigned long   u32;
sfr       P3M1    =       0xB1;           //P3M1.n,P3M0.n   =00--->Standard,   01--->push-pull
```

STC MCU Limited

```
sfr     P3M0    =       0xB2;           //              =10--->pure input,  11--->open drain
sfr     AUXR    =       0x8E;
sfr     INT_CLKO=       0x8F;
sbit    P_PWM   =       P3^4;           //定义PWM输出引脚。

u16     pwm;                    //定义PWM输出高电平的时间的变量。用户操作PWM的变量。
u16     PWM_high, PWM_low;      //中间变量，用户请勿修改。

void    delay_ms(u8 ms);
void    LoadPWM(u16 i);

/********************* 主函数**************************/
void main(void)
{
        P_PWM = 0;
        P3M1 &= ~(1 << 4);          //P3.4 设置为推挽输出
        P3M0 |=  (1 << 4);

        TR1 = 0;                    //停止计数
        ET1 = 1;                    //允许中断
        PT1 = 1;                    //高优先级中断
        TMOD &= ~0x30;              //工作模式,0: 16位自动重装
        AUXR |=  0x40;              //1T
        TMOD &= ~0x40;              //定时
        INT_CLKO |=  0x02;          //输出时钟

        TH1 = 0;
        TL1 = 0;
        TR1 = 1;                    //开始运行

        EA = 1;

        pwm = PWM_DUTY / 10;    //给PWM一个初值，这里为10%占空比
        LoadPWM(pwm);           //计算PWM重装值

        while (1)
        {
                while(pwm < (PWM_HIGH_MAX-8))
                {
                        pwm += 8;                   //PWM逐渐加到最大
                        LoadPWM(pwm);
                        delay_ms(8);
                }
                while(pwm > (PWM_HIGH_MIN+8))
                {
                        pwm -= 8;           //PWM逐渐减到最小
                        LoadPWM(pwm);
                        delay_ms(8);
                }
```

STC MCU Limited

```
        }
}
//========================================================================
// 函数: void  delay_ms(u8 ms)
// 描述: 延时函数。
// 参数: ms,要延时的ms数, 这里只支持1~255ms. 自动适应主时钟.
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//========================================================================
void  delay_ms(u8 ms)
{
        unsigned int i;
         do{
                i = MAIN_Fosc / 13000;
                  while(--i)            ;
        }while(--ms);
}
/****************** 计算PWM重装值函数 ********************/
void    LoadPWM(u16 i)
{
        u16     j;

        if(i > PWM_HIGH_MAX)    i = PWM_HIGH_MAX;
                                        //如果写入大于最大占空比数据，则强制为最大占空比。
        if(i < PWM_HIGH_MIN)    i = PWM_HIGH_MIN;
                                        //如果写入小于最小占空比数据，则强制为最小占空比。
        j = 65536UL - PWM_DUTY + i;     //计算PWM低电平时间
        i = 65536UL - i;                //计算PWM高电平时间
        EA = 0;
        PWM_high = i;                   //装载PWM高电平时间
        PWM_low  = j;                   //装载PWM低电平时间
        EA = 1;
}
/********************** Timer0中断函数************************/
void timer0_int (void) interrupt 3
{
        if(P_PWM)
        {
                TH1 = (u8)(PWM_low >> 8);           //如果是输出高电平，则装载低电平时间。
                TL1 = (u8)PWM_low;
        }
        else
        {
                TH1 = (u8)(PWM_high >> 8);          //如果是输出低电平，则装载高电平时间。
                TL1 = (u8)PWM_high;
        }
}
```

STC MCU Limited

2、汇编语言程序

```
;/*---------------------------------------------------------------------------------------------------*/
;/* --- STC MCU Limited ---------------------------------------------------------------------------*/
;/* --- Mobile: (86)13922805190 -------------------------------------------------------------------*/
;/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
;/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
;/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
;/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
;/*---------------------------------------------------------------------------------------------------*/

;/*************  功能说明            **************
;本程序演示使用定时器做软件PWM。

;定时器1做16位自动重装，中断，从T1CLKO高速输出PWM。

;本例程是使用STC15F/L系列MCU的定时器T1做模拟PWM的例程。

;PWM可以是任意的量程。但是由于软件重装需要一点时间，所以PWM占空比最小为32T/周期，最大为(周期-32T)/周期, T为时钟周期。

;PWM频率为周期的倒数。假如周期为6000, 使用24MHZ的主频，则PWM频率为4000HZ。

;*****************************************/
;*****************************用户宏定义*****************************************************************
Fosc_KHZ        EQU     24000   //定义主时钟, KHZ

PWM_DUTY        EQU     6000    //定义PWM的周期，数值为PCA所选择的时钟脉冲个数。
PWM_HIGH_MIN    EQU     32      //限制PWM输出的最小占空比, 避免中断里重装参数时间不够。
PWM_HIGH_MAX    EQU     (PWM_DUTY - PWM_HIGH_MIN)   //限制PWM输出的最大占空比。

;*********************************************************************************************************

P3M1            DATA    0B1H    ; P3M1.n,P3M0.n   =00--->Standard,    01--->push-pull
P3M0            DATA    0B2H    ;                 =10--->pure input,  11--->open drain
AUXR            DATA    08EH    ;
INT_CLKO        DATA    08FH    ;

P_PWM  BIT      P3.4            ; 定义PWM输出引脚。

pwm_H           DATA    030H    ; 定义PWM输出高电平的时间的变量。用户操作PWM的变量。
pwm_L           DATA    031H
PWM_high_H      DATA    032H    ; 中间变量，用户请勿修改
PWM_high_L      DATA    033H
PWM_low_H       DATA    034H
PWM_low_L       DATA    035H

STACK_POIRTER   EQU             0D0H    ;堆栈开始地址
;*************************************************************************
;*************************************************************************
```

STC MCU Limited

```
        ORG     00H                             ;reset
        LJMP    F_Main

        ORG     1BH                             ;3  Timer1 interrupt
        LJMP    F_Timer1_Interrupt

;********************* 主程序 ***************************/
F_Main:
        MOV     SP, #STACK_POIRTER
        MOV     PSW, #0
        USING   0                               ;选择第0组R0~R7
;================= 用户初始化程序 ====================================
        CLR     P_PWM
        ANL     P3M1, #NOT (1 SHL 4)            ; P3.5 设置为推挽输出
        ORL     P3M0, #(1 SHL 4);

        CLR     TR1                             ; 停止计数
        SETB    ET1                             ; 允许中断
        SETB    PT1                             ; 高优先级中断
        ANL     TMOD, #NOT 030H                 ; 工作模式,0: 16位自动重装
        ORL     AUXR, #040H                     ; 1T
        ANL     TMOD, #NOT 040H                 ; 定时
        ORL     INT_CLKO, #002H                 ; 输出时钟

        MOV     TH1, #0
        MOV     TL1, #0
        SETB    TR1                             ; 开始运行

        SETB    EA

        MOV     pwm_H, #HIGH (PWM_DUTY / 10)    ;给PWM一个初值，这里为10%占空比
        MOV     pwm_L, #LOW  (PWM_DUTY / 10)
        MOV     R6, pwm_H
        MOV     R7, pwm_L
        LCALL   F_PWMn_Update                   ; 计算PWM重装值
;=================== 主循环 ==================================
L_MainLoop1:
        MOV     R7, #2
        LCALL   F_delay_ms

        MOV     A, pwm_L                        ; if(++pwm >= PWM_HIGH_MAX)
        ADD     A, #1
        MOV             pwm_L, A
        MOV     A, pwm_H
        ADDC    A, #0
        MOV     pwm_H, A
        MOV     A, pwm_L
        CLR     C
        SUBB    A, #LOW PWM_HIGH_MAX
```

STC MCU Limited

```
        MOV     A, pwm_H
        SUBB    A, #HIGH PWM_HIGH_MAX
        JC      L_PWM_NotUpOverFollow           ; PWM逐渐加到最大
        MOV     pwm_H, #HIGH PWM_HIGH_MAX
        MOV     pwm_L, #LOW PWM_HIGH_MAX
        SJMP    L_MainLoop2
L_PWM_NotUpOverFollow:
        MOV     R6, pwm_H
        MOV     R7, pwm_L
        LCALL   F_PWMn_Update
        SJMP    L_MainLoop1

L_MainLoop2:
        MOV     R7, #2
        LCALL   F_delay_ms

        MOV     A, pwm_L                        ; if(++pwm < PWM_HIGH_MIN)
        CLR     C
        SUBB    A, #1
        MOV     pwm_L, A
        MOV     A, pwm_H
        SUBB    A, #0
        MOV     pwm_H, A
        MOV     A, pwm_L
        CLR     C
        SUBB    A, #LOW PWM_HIGH_MIN
        MOV     A, pwm_H
        SUBB    A, #HIGH PWM_HIGH_MIN
        JNC     L_PWM_NotDnOverFollow           ; PWM逐渐减到最小
        MOV     pwm_H, #HIGH PWM_HIGH_MIN
        MOV     pwm_L, #LOW PWM_HIGH_MIN
        SJMP    L_MainLoop1
L_PWM_NotDnOverFollow:
        MOV     R6, pwm_H
        MOV     R7, pwm_L
        LCALL   F_PWMn_Update
        SJMP    L_MainLoop2
;//==========================================================================
;// 函数: F_delay_ms
;// 描述: 延时子程序。
;// 参数: R7: 延时ms数.
;// 返回: none.
;// 版本: VER1.0
;// 日期: 2013-4-1
;// 备注: 除了ACCC和PSW外, 所用到的通用寄存器都入栈
;//==========================================================================
F_delay_ms:
        PUSH    AR3                             ;入栈R3
        PUSH    AR4                             ;入栈R4
```

STC MCU Limited

```
L_delay_ms_1:
        MOV     R3, #HIGH (Fosc_KHZ / 13)
        MOV     R4, #LOW (Fosc_KHZ / 13)

L_delay_ms_2:
        MOV     A, R4                           ;1T             Total 13T/loop
        DEC     R4                              ;2T
        JNZ     L_delay_ms_3                    ;4T
        DEC     R3
L_delay_ms_3:
        DEC     A                               ;1T
        ORL     A, R3                           ;1T
        JNZ     L_delay_ms_2                    ;4T

        DJNZ    R7, L_delay_ms_1

        POP     AR4                             ;出栈R2
        POP     AR3                             ;出栈R3
        RET
;========================================================================
; 函数: F_PWMn_Update
; 描述: 更新占空比数据。
; 参数: R6,R7: PWM值。
; 返回: 无
; 版本: VER1.0
; 日期: 2014-2-15
; 备注:
;========================================================================
F_PWMn_Update:
        PUSH    AR3
        PUSH    AR4

        CLR     C
        MOV     A, R7
        SUBB    A, #LOW PWM_HIGH_MAX
        MOV     A, R6
        SUBB    A, #HIGH PWM_HIGH_MAX
        JC      L_QuitCheckPwm_1
        MOV     R6, #HIGH PWM_HIGH_MAX
                                        ; 如果写入大于最大占空比数据，强制为最大占空比。
        MOV     R7, #LOW  PWM_HIGH_MAX
L_QuitCheckPwm_1:
        CLR     C
        MOV     A, R7
        SUBB    A, #LOW PWM_HIGH_MIN
        MOV     A, R6
        SUBB    A, #HIGH PWM_HIGH_MIN
        JNC     L_QuitCheckPwm_2
        MOV     R6, #HIGH PWM_HIGH_MIN
```

STC MCU Limited

```
                                        ; 如果写入小于最小占空比数据，强制为最小占空比。
        MOV     R7, #LOW  PWM_HIGH_MIN
L_QuitCheckPwm_2:

        CLR     C
        MOV     A, R7                   ;计算并保存PWM输出低电平的T0时钟脉冲个数
        SUBB    A, #LOW PWM_DUTY
        MOV     R4, A
        MOV     A, R6
        SUBB    A, #HIGH PWM_DUTY
        MOV     R3, A

        CLR     C
        MOV     A, #0                   ;计算并保存PWM输出高电平的T0时钟脉冲个数
        SUBB    A, R7
        MOV     R7, A
        MOV     A, #0
        SUBB    A, R6
        MOV     R6, A

        CLR     EA                      ; 禁止一会中断，一般不会影响PWM。
        MOV     PWM_high_H, R6          ; 数据装入占空比变量。
        MOV     PWM_high_L, R7
        MOV     PWM_low_H, R3
        MOV     PWM_low_L, R4
        SETB    EA

        POP     AR4
        POP     AR3
        RET
;********************** Timer0中断函数************************/
F_Timer1_Interrupt:
        PUSH    PSW
        PUSH    ACC
        JNB     P_PWM, L_T1_LoadLow
        MOV     TH1, PWM_low_H          ; 如果是输出高电平，则装载低电平时间。
        MOV     TL1, PWM_low_L
        SJMP    L_QuitTimer1
L_T1_LoadLow:
        MOV     TH1, PWM_high_H                 ; 如果是输出低电平，则装载高电平时间。
        MOV     TL1, PWM_high_L

L_QuitTimer1:
        POP     ACC
        POP     PSW

        RETI

        END
```

STC MCU Limited

11.13 用T2的时钟输出功能实现8~16位PWM的程序(C及汇编)——占用系统时间小于0.4%

下文为利用T2的时钟输出功能实现8~16位PWM（占用系统时间小于0.4%）的测试程序（包括C语言程序和汇编程序）。

1、C语言程序

```
/*---------------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------------*/
/* --- STC 1T Series MCU RC Demo -------------------------------------------*/
/* --- Mobile: (86)13922805190 ---------------------------------------------*/
/* --- Fax: 86-0513-55012956,55012947,55012969 -----------------------------*/
/* --- Tel: 86-0513-55012928,55012929,55012966 -----------------------------*/
/* --- Web: www.GXWMCU.com -------------------------------------------------*/
/* --- QQ:  800003751 ------------------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------------*/

#include  <reg52.h>

/*************    功能说明    ***************
本程序演示使用定时器做软件PWM。

定时器2做16位自动重装，中断，从T2CLKO高速输出PWM。

本例程是使用STC15F/L系列MCU的定时器T2做模拟PWM的例程。

PWM可以是任意的量程。但是由于软件重装需要一点时间，所以PWM占空比最小为32T/周期，最大为(周期-32T)/周期, T为时钟周期。

PWM频率为周期的倒数。假如周期为6000, 使用24MHZ的主频，则PWM频率为4000HZ。
*****************************************/
#define   MAIN_Fosc       24000000UL      //定义主时钟

#define   PWM_DUTY        6000            //定义PWM的周期，数值为时钟周期数，假如使用
                                          //24.576MHZ的主频，则PWM频率为6000HZ。

#define   PWM_HIGH_MIN 32                 //限制PWM输出的最小占空比。用户请勿修改。
#define   PWM_HIGH_MAX(PWM_DUTY-PWM_HIGH_MIN)
                                          //限制PWM输出的最大占空比。用户请勿修改。
typedef   unsigned char   u8;
typedef   unsigned int    u16;
typedef   unsigned long   u32;
sfr       IE2     =       0xAF;           //STC12C5A60S2系列
sfr       P3M1    =       0xB1;           //P3M1.n,P3M0.n   =00--->Standard,   01--->push-pull
sfr       P3M0    =       0xB2;           //                =10--->pure input, 11--->open drain
```

STC MCU Limited

```
sfr     AUXR    =       0x8E;
sfr     INT_CLKO  =     0x8F;

sbit    P_PWM   =       P3^0;           //定义PWM输出引脚。

u16     pwm;                    //定义PWM输出高电平的时间的变量。用户操作PWM的变量。
u16     PWM_high,PWM_low;       //中间变量，用户请勿修改。

void    delay_ms(u8 ms);
void    LoadPWM(u16 i);

/********************* 主函数**************************/
void main(void)
{
        P_PWM = 0;
        P3M1 &= ~1;             //P3.0 设置为推挽输出
        P3M0 |=  1;

        AUXR &= ~(1<<4);        //停止计数
        IE2  |=  (1<<2);        //允许中断
        AUXR |=  (1<<2);  //1T
        AUXR &= ~(1<<3);        //定时
        INT_CLKO |=  0x04;      //输出时钟

        TH2 = 0;
        TL2 = 0;
        AUXR |=  (1<<4);        //开始运行

        EA = 1;

        pwm = PWM_DUTY / 10;    //给PWM一个初值，这里为10%占空比
        LoadPWM(pwm);           //计算PWM重装值

        while (1)
        {
                while(pwm < (PWM_HIGH_MAX-8))
                {
                        pwm += 8;               //PWM逐渐加到最大
                        LoadPWM(pwm);
                        delay_ms(8);
                }
                while(pwm > (PWM_HIGH_MIN+8))
                {
                        pwm -= 8;               //PWM逐渐减到最小
                        LoadPWM(pwm);
                        delay_ms(8);
                }
        }
}
```

STC MCU Limited

```
//========================================================================
// 函数: void  delay_ms(u8 ms)
// 描述: 延时函数。
// 参数: ms,要延时的ms数, 这里只支持1~255ms. 自动适应主时钟.
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//========================================================================
void  delay_ms(u8 ms)
{
    unsigned int i;
         do{
                 i = MAIN_Fosc / 13000;
                  while(--i)          ;
         }while(--ms);
}

/****************** 计算PWM重装值函数 ********************/
void    LoadPWM(u16 i)
{
         u16    j;

         if(i > PWM_HIGH_MAX)    i = PWM_HIGH_MAX;
                                 //如果写入大于最大占空比数据，则强制为最大占空比。
         if(i < PWM_HIGH_MIN)    i = PWM_HIGH_MIN;
                                 //如果写入小于最小占空比数据，则强制为最小占空比。
         j = 65536UL - PWM_DUTY + i;     //计算PWM低电平时间
         i = 65536UL - i;                //计算PWM高电平时间
         EA = 0;
         PWM_high = i;                   //装载PWM高电平时间
         PWM_low  = j;                   //装载PWM低电平时间
         EA = 1;
}

/********************** Timer0中断函数************************/
void timer2_int (void) interrupt 12
{
         if(P_PWM)
         {
                 TH2 = (u8)(PWM_low >> 8);          //如果是输出高电平，则装载低电平时间。
                 TL2 = (u8)PWM_low;
         }
         else
         {
                 TH2 = (u8)(PWM_high >> 8);         //如果是输出低电平，则装载高电平时间。
                 TL2 = (u8)PWM_high;
         }
}
```

STC MCU Limited

2、汇编语言程序

```
;/*---------------------------------------------------------------------------------------------------*/
;/* --- STC MCU Limited ------------------------------------------------------------------------------*/
;/* --- Mobile: (86)13922805190 ----------------------------------------------------------------------*/
;/* --- Fax: 86-755-82905966 -------------------------------------------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------------------------------------------*/
;/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
;/*---------------------------------------------------------------------------------------------------*/

;/**************  功能说明              ***************
;本程序演示使用定时器做软件PWM。

;定时器2做16位自动重装，中断，从T2CLKO高速输出PWM。

;本例程是使用STC15F/L系列MCU的定时器T2做模拟PWM的例程。

;PWM可以是任意的量程。但是由于软件重装需要一点时间，所以PWM占空比最小为32T/周期，最大为(周期-32T)/周期, T为时钟周期。

;PWM频率为周期的倒数。假如周期为6000, 使用24MHZ的主频，则PWM频率为4000HZ。

;*****************************************/
;*****************************用户宏定义*****************************************************
Fosc_KHZ        EQU     24000   //定义主时钟, KHZ

PWM_DUTY        EQU     6000    //定义PWM的周期，数值为PCA所选择的时钟脉冲个数。
PWM_HIGH_MIN    EQU     32      //限制PWM输出的最小占空比, 避免中断里重装参数时间不够。
PWM_HIGH_MAXEQU (PWM_DUTY - PWM_HIGH_MIN)
                        //限制PWM输出的最大占空比。
;*********************************************************************************************
P3M1            DATA    0B1H            ; P3M1.n,P3M0.n   =00--->Standard,    01--->push-pull
P3M0            DATA    0B2H            ;                 =10--->pure input,  11--->open drain
AUXR            DATA    08EH            ;
INT_CLKO        DATA    08FH            ;
IE2             DATA    0AFH
T2H             DATA    0D6H
T2L             DATA    0D7H
P_PWM  BIT      P3.0            ; 定义PWM输出引脚。

pwm_H           DATA    030H    ; 定义PWM输出高电平的时间的变量。用户操作PWM的变量。
pwm_L           DATA    031H
PWM_high_H      DATA    032H    ; 中间变量，用户请勿修改
PWM_high_L      DATA    033H
PWM_low_H       DATA    034H
PWM_low_L       DATA    035H

STACK_POIRTER EQU       0D0H    ;堆栈开始地址
;*************************************************************************
```

STC MCU Limited

```
;*************************************************************************
        ORG     00H                     ;reset
        LJMP    F_Main

        ORG     63H                     ;12  Timer2 interrupt
        LJMP    F_Timer2_Interrupt
;******************** 主程序 **************************/
F_Main:
        MOV     SP, #STACK_POIRTER
        MOV     PSW, #0
        USING   0                       ;选择第0组R0~R7
;================ 用户初始化程序 ====================================
        CLR     P_PWM
        ANL     P3M1, #NOT 1            ; P3.0 设置为推挽输出
        ORL     P3M0, #1

        ANL     AUXR, #NOT (1 SHL 4)    ;停止计数
        ORL     IE2, #(1 SHL 2)         ; 允许中断
        ORL     AUXR, #(1 SHL 2)        ; 1T
        ANL     AUXR, #NOT (1 SHL 3)    ; 定时
        ORL     INT_CLKO, #0x04         ; 输出时钟

        MOV     T2H, #0;
        MOV     T2L, #0;
        ORL     AUXR, #(1 SHL 4)        ; 开始运行

        SETB    EA

        MOV     pwm_H, #HIGH (PWM_DUTY / 10)        ;给PWM一个初值，这里为10%占空比
        MOV     pwm_L, #LOW  (PWM_DUTY / 10)
        MOV     R6, pwm_H
        MOV     R7, pwm_L
        LCALL   F_PWMn_Update                       ; 计算PWM重装值

;=================== 主循环 ==================================
L_MainLoop1:
        MOV     R7, #2
        LCALL   F_delay_ms

        MOV     A, pwm_L                            ; if(++pwm >= PWM_HIGH_MAX)
        ADD     A, #1
        MOV     pwm_L, A
        MOV     A, pwm_H
        ADDC    A, #0
        MOV     pwm_H, A
        MOV     A, pwm_L
        CLR     C
        SUBB    A, #LOW PWM_HIGH_MAX
        MOV     A, pwm_H
```

STC MCU Limited

```
        SUBB    A, #HIGH PWM_HIGH_MAX
        JC      L_PWM_NotUpOverFollow           ; PWM逐渐加到最大
        MOV     pwm_H, #HIGH PWM_HIGH_MAX
        MOV     pwm_L, #LOW PWM_HIGH_MAX
        SJMP    L_MainLoop2
L_PWM_NotUpOverFollow:
        MOV     R6, pwm_H
        MOV     R7, pwm_L
        LCALL   F_PWMn_Update
        SJMP    L_MainLoop1

L_MainLoop2:
        MOV     R7, #2
        LCALL   F_delay_ms

        MOV     A, pwm_L                        ; if(++pwm < PWM_HIGH_MIN)
        CLR     C
        SUBB    A, #1
        MOV     pwm_L, A
        MOV     A, pwm_H
        SUBB    A, #0
        MOV     pwm_H, A
        MOV     A, pwm_L
        CLR     C
        SUBB    A, #LOW PWM_HIGH_MIN
        MOV     A, pwm_H
        SUBB    A, #HIGH PWM_HIGH_MIN
        JNC     L_PWM_NotDnOverFollow           ; PWM逐渐减到最小
        MOV     pwm_H, #HIGH PWM_HIGH_MIN
        MOV     pwm_L, #LOW PWM_HIGH_MIN
        SJMP    L_MainLoop1
L_PWM_NotDnOverFollow:
        MOV     R6, pwm_H
        MOV     R7, pwm_L
        LCALL   F_PWMn_Update
        SJMP    L_MainLoop2
;//========================================================================
;// 函数: F_delay_ms
;// 描述: 延时子程序。
;// 参数: R7: 延时ms数.
;// 返回: none.
;// 版本: VER1.0
;// 日期: 2013-4-1
;// 备注: 除了ACCC和PSW外, 所用到的通用寄存器都入栈
;//========================================================================
F_delay_ms:
        PUSH    AR3                             ;入栈R3
        PUSH    AR4                             ;入栈R4
```

STC MCU Limited

```
L_delay_ms_1:
        MOV     R3, #HIGH (Fosc_KHZ / 13)
        MOV     R4, #LOW (Fosc_KHZ / 13)

L_delay_ms_2:
        MOV     A, R4                           ;1T             Total 13T/loop
        DEC     R4                              ;2T
        JNZ     L_delay_ms_3                    ;4T
        DEC     R3
L_delay_ms_3:
        DEC     A                               ;1T
        ORL     A, R3                           ;1T
        JNZ     L_delay_ms_2                    ;4T

        DJNZ    R7, L_delay_ms_1

        POP     AR4                             ;出栈R2
        POP     AR3                             ;出栈R3
        RET
;=========================================================================
; 函数: F_PWMn_Update
; 描述: 更新占空比数据。
; 参数: R6,R7: PWM值。
; 返回: 无
; 版本: VER1.0
; 日期: 2014-2-15
; 备注:
;=========================================================================
F_PWMn_Update:
        PUSH    AR3
        PUSH    AR4

        CLR     C
        MOV     A, R7
        SUBB    A, #LOW PWM_HIGH_MAX
        MOV     A, R6
        SUBB    A, #HIGH PWM_HIGH_MAX
        JC      L_QuitCheckPwm_1
        MOV     R6, #HIGH PWM_HIGH_MAX
                                        ; 如果写入大于最大占空比数据，强制为最大占空比。
        MOV     R7, #LOW  PWM_HIGH_MAX
L_QuitCheckPwm_1:

        CLR     C
        MOV     A, R7
        SUBB    A, #LOW PWM_HIGH_MIN
        MOV     A, R6
        SUBB    A, #HIGH PWM_HIGH_MIN
        JNC     L_QuitCheckPwm_2
```

STC MCU Limited

```
        MOV     R6, #HIGH PWM_HIGH_MIN
                                        ; 如果写入小于最小占空比数据，强制为最小占空比。
        MOV     R7, #LOW  PWM_HIGH_MIN
L_QuitCheckPwm_2:

        CLR     C
        MOV     A, R7                   ;计算并保存PWM输出低电平的T0时钟脉冲个数
        SUBB    A, #LOW PWM_DUTY
        MOV     R4, A
        MOV     A, R6
        SUBB    A, #HIGH PWM_DUTY
        MOV     R3, A

        CLR     C
        MOV     A, #0                   ;计算并保存PWM输出高电平的T0时钟脉冲个数
        SUBB    A, R7
        MOV     R7, A
        MOV     A, #0
        SUBB    A, R6
        MOV     R6, A

        CLR     EA                      ; 禁止一会中断， 一般不会影响PWM。
        MOV     PWM_high_H, R6          ; 数据装入占空比变量。
        MOV     PWM_high_L, R7
        MOV     PWM_low_H, R3
        MOV     PWM_low_L, R4
        SETB    EA

        POP     AR4
        POP     AR3
        RET

;********************** Timer0中断函数************************/
F_Timer2_Interrupt:
        PUSH    PSW
        PUSH    ACC
        JNB     P_PWM, L_T2_LoadLow
        MOV     T2H, PWM_low_H          ; 如果是输出高电平，则装载低电平时间。
        MOV     T2L, PWM_low_L
        SJMP    L_QuitTimer2
L_T2_LoadLow:
        MOV     T2H, PWM_high_H         ; 如果是输出低电平，则装载高电平时间。
        MOV     T2L, PWM_high_L

L_QuitTimer2:
        POP             ACC
        POP             PSW
        RETI
        END
```

STC MCU Limited

11.14 利用两路CCP/PCA模拟一个全双工串口的程序(C及汇编)

1. C程序：

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------*/

/*************    功能说明            ***************

使用STC15系列的PCA0和PCA1做的模拟串口. PCA0接收(P2.5), PCA1发送(P2.6).

假定测试芯片的工作频率为22118400Hz. 时钟为5.5296MHZ ~ 35MHZ.

波特率高，则时钟也要选高, 优先使用 22.1184MHZ, 11.0592MHZ.

测试方法: 上位机发送数据,MCU收到数据后原样返回.

串口固定设置: 1位起始位, 8位数据位, 1位停止位, 波特率在 600 ~ 57600 bps.

1200 ~ 57600 bps @ 33.1776MHZ
 600 ~ 57600 bps @ 22.1184MHZ
 600 ~ 38400 bps @ 18.4320MHZ
 300 ~ 28800 bps @ 11.0592MHZ
 150 ~ 14400 bps @  5.5296MHZ

*****************************************/
#include  <reg52.h>

#define MAIN_Fosc               22118400UL          //定义主时钟
#define   UART3_Baudrate        57600UL             //定义波特率
#define   RX_Lenth              16                  //接收长度

#define   PCA_P12_P11_P10_P37   (0<<4)
#define   PCA_P34_P35_P36_P37   (1<<4)
#define   PCA_P24_P25_P26_P27   (2<<4)
#define   PCA_Mode_Capture      0
#define   PCA_Mode_SoftTimer    0x48
#define   PCA_Clock_1T          (4<<1)
#define   PCA_Clock_2T          (1<<1)
#define   PCA_Clock_4T          (5<<1)
#define   PCA_Clock_6T          (6<<1)
#define   PCA_Clock_8T          (7<<1)
#define   PCA_Clock_12T         (0<<1)
```

STC MCU Limited

```
#define   PCA_Clock_ECI          (3<<1)
#define   PCA_Rise_Active        (1<<5)
#define   PCA_Fall_Active        (1<<4)
#define   PCA_PWM_8bit           (0<<6)
#define   PCA_PWM_7bit           (1<<6)
#define   PCA_PWM_6bit           (2<<6)

#define   UART3_BitTime   (MAIN_Fosc / UART3_Baudrate)

#define   ENABLE          1
#define   DISABLE         0

typedef   unsigned char   u8;
typedef   unsigned int    u16;
typedef   unsigned long   u32;

sfr       AUXR1  =        0xA2;
sfr       CCON   =        0xD8;
sfr       CMOD   =        0xD9;
sfr       CCAPM0=         0xDA;          //PCA模块0的工作模式寄存器。
sfr       CCAPM  =        0xDB;          //PCA模块1的工作模式寄存器。
sfr       CCAPM2=         0xDC;          //PCA模块2的工作模式寄存器。

sfr       CL     =        0xE9;
sfr       CCAP0L =        0xEA;          //PCA模块0的捕捉/比较寄存器低8位。
sfr       CCAP1L =        0xEB;          //PCA模块1的捕捉/比较寄存器低8位。
sfr       CCAP2L =        0xEC;          //PCA模块2的捕捉/比较寄存器低8位。

sfr       CH     =        0xF9;
sfr       CCAP0H =        0xFA;          //PCA模块0的捕捉/比较寄存器高8位。
sfr       CCAP1H =        0xFB;          //PCA模块1的捕捉/比较寄存器高8位。
sfr       CCAP2H =        0xFC;          //PCA模块2的捕捉/比较寄存器高8位。

sbit      CCF0   =        CCON^0;        //PCA 模块0中断标志，由硬件置位，必须由软件清0。
sbit      CCF1   =        CCON^1;        //PCA 模块1中断标志，由硬件置位，必须由软件清0。
sbit      CCF2   =        CCON^2;        //PCA 模块2中断标志，由硬件置位，必须由软件清0。
sbit      CR     =        CCON^6;        //1: 允许PCA计数器计数，0: 禁止计数。
sbit      CF     =        CCON^7;        //PCA计数器溢出（CH，CL由FFFFH变为0000H）标志。
                                         //PCA计数器溢出后由硬件置位，必须由软件清0。
sbit      PPCA   =        IP^7;          //PCA 中断 优先级设定位
u16       CCAP0_tmp;
u16       CCAP1_tmp;
u8        Tx3_read;                      //发送读指针
u8        Rx3_write;                     //接收写指针
u8 idata  buf3[RX_Lenth];                //接收缓冲

//==================== 模拟串口相关==========================
sbit      P_RX3 = P2^5;           //定义模拟串口接收IO
sbit      P_TX3 = P2^6;           //定义模拟串口发送IO
```

STC MCU Limited

```
u8      Tx3_DAT;                    // 发送移位变量, 用户不可见
u8      Rx3_DAT;                    // 接收移位变量, 用户不可见
u8      Tx3_BitCnt;                 // 发送数据的位计数器, 用户不可见
u       Rx3_BitCnt;                 // 接收数据的位计数器, 用户不可见
u8      Rx3_BUF;                    // 接收到的字节, 用户读取
u8      Tx3_BUF;                    // 要发送的字节, 用户写入
bit     Rx3_Ring;                   // 正在接收标志, 低层程序使用, 用户程序不可见
bit     Tx3_Ting;                   // 正在发送标志, 用户置1请求发送, 底层发送完成清0
bit     RX3_End;                    // 接收到一个字节, 用户查询 并清0
//=============================================================
void    PCA_Init(void);
/********************* 主函数 **************************/
void main(void)
{
        PCA_Init();                                     //PCA初始化
        EA = 1;

        Tx3_read  = 0;
        Rx3_write = 0;
        Tx3_Ting  = 0;
        Rx3_Ring  = 0;
        RX3_End   = 0;
        Tx3_BitCnt = 0;
        while (1)                                       //user's function
        {
                if (RX3_End)                            // 检测是否收到一个字节
                {
                        RX3_End = 0;                    // 清除标志
                        buf3[Rx3_write] = Rx3_BUF;      // 写入缓冲
                        if(++Rx3_write >= RX_Lenth)     Rx3_write = 0;
                                                        // 指向下一个位置, 溢出检测
                }
                if (!Tx3_Ting)                          // 检测是否发送空闲
                {
                        if (Tx3_read != Rx3_write)      // 检测是否收到过字符
                        {
                                Tx3_BUF = buf3[Tx3_read]; // 从缓冲读一个字符发送
                                Tx3_Ting = 1;           // 设置发送标志
                                if(++Tx3_read >= RX_Lenth)      Tx3_read = 0;
                                                        // 指向下一个位置, 溢出检测
                        }
                }
        }
}
//========================================================================
// 函数: void      PCA_Init(void)
// 描述: PCA初始化程序.
// 参数: none
// 返回: none.
```

STC MCU Limited

```
// 版本: V1.0, 2013-11-22
//========================================================================
void    PCA_Init(void)
{
    CR = 0;
    CCAPM0 = (PCA_Mode_Capture | PCA_Fall_Active | ENABLE);   //16位下降沿捕捉中断模式

    CCAPM1    = PCA_Mode_SoftTimer | ENABLE;
    CCAP1_tmp = UART3_BitTime;
    CCAP1L    = (u8)CCAP1_tmp;                  //将影射寄存器写入捕获寄存器，先写CCAP0L
    CCAP1H    = (u8)(CCAP1_tmp >> 8);           //后写CCAP0H

    CH = 0;
    CL = 0;
    AUXR1 = (AUXR1 & ~(3<<4)) | PCA_P24_P25_P26_P27;                    //切换IO口
    CMOD  = (CMOD  & ~(7<<1)) | PCA_Clock_1T;                           //选择时钟源
    PPCA  = 1;                                                          // 高优先级中断
    CR    = 1;                                                          // 运行PCA定时器
}
//========================================================================
//========================================================================
// 函数: void        PCA_Handler (void) interrupt 7
// 描述: PCA中断处理程序.
// 参数: None
// 返回: none.
// 版本: V1.0, 2012-11-22
//========================================================================
void    PCA_Handler (void) interrupt 7
{
    if(CCF0)                                                            //PCA模块0中断
    {
        CCF0 = 0;                                                       //清PCA模块0中断标志
        if(Rx3_Ring)                                                    //已收到起始位
        {
            if (--Rx3_BitCnt == 0)                                      //接收完一帧数据
            {
                Rx3_Ring = 0;                                           //停止接收
                Rx3_BUF = Rx3_DAT;                                      //存储数据到缓冲区
                RX3_End = 1;
                CCAPM0 = (PCA_Mode_Capture | PCA_Fall_Active | ENABLE);
                                                                        //16位下降沿捕捉中断模式
            }
            else
            {
                Rx3_DAT >>= 1;   //把接收的单b数据 暂存到 RxShiftReg(接收缓冲)
                if(P_RX3) Rx3_DAT |= 0x80;                              //shift RX data to RX buffer
                CCAP0_tmp += UART3_BitTime;                             //数据位时间
                CCAP0L = (u8)CCAP0_tmp;                                 //将影射寄存器写入捕获寄存
                                                                        //器，先写CCAP0L
```

STC MCU Limited

```
                    CCAP0H = (u8)(CCAP0_tmp >> 8);    //后写CCAP0H
                }
            }
            else
            {
                CCAP0_tmp = ((u16)CCAP0H << 8) + CCAP0L;          //读捕捉寄存器
                CCAP0_tmp += (UART3_BitTime / 2 + UART3_BitTime);//起始位 + 半个数据位
                CCAP0L = (u8)CCAP0_tmp;                   //将影射寄存器写入捕获
                                                          //寄存器，先写CCAP0L
                CCAP0H = (u8)(CCAP0_tmp >> 8);            //后写CCAP0H
                CCAPM0 = (PCA_Mode_SoftTimer | ENABLE);//16位软件定时中断模式
                Rx3_Ring = 1;                             //标志已收到起始位
                Rx3_BitCnt = 9;                           //初始化接收的数据位数(8个数
                                                          //据位+1个停止位)
            }
        }

        if(CCF1) //PCA模块1中断, 16位软件定时中断模式
        {
            CCF1 = 0;                                     //清PCA模块1中断标志
            CCAP1_tmp += UART3_BitTime;
            CCAP1L = (u8)CCAP1_tmp;           //将影射寄存器写入捕获寄存器，先写CCAP0L
            CCAP1H = (u8)(CCAP1_tmp >> 8);    //后写CCAP0H

            if(Tx3_Ting)                      // 不发送, 退出
            {
                if(Tx3_BitCnt == 0)           //发送计数器为0 表明单字节发送还没开始
                {
                    P_TX3 = 0;                //发送开始位
                    Tx3_DAT = Tx3_BUF;        //把缓冲的数据放到发送的buff
                    Tx3_BitCnt = 9;           //发送数据位数 (8数据位+1停止位)
                }
                else                          //发送计数器为非0 正在发送数据
                {
                    if (--Tx3_BitCnt == 0)    //发送计数器减为0 表明单字节发送结束
                    {
                        P_TX3 = 1;            //送停止位数据
                        Tx3_Ting = 0;         //发送停止
                    }
                    else
                    {
                        Tx3_DAT >>= 1;        //把最低位送到 CY(益处标志位)
                        P_TX3 = CY;           //发送一个bit数据
                    }
                }
            }
        }
}
```

STC MCU Limited

2. 汇编程序:

```
;/*---------------------------------------------------------------------------------------------*/
;/* --- STC MCU Limited ---------------------------------------------------------------------*/
;/* --- Mobile: (86)13922805190 -------------------------------------------------------------*/
;/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
;/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
;/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
;/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
;/*---------------------------------------------------------------------------------------------*/

;*************     功能说明              ***************
;使用STC15系列的PCA0和PCA1做的模拟串口. PCA0接收(P2.5), PCA1发送(P2.6).

;假定测试芯片的工作频率为22118400Hz. 时钟为5.5296MHZ ~ 35MHZ.

;波特率高，则时钟也要选高, 优先使用 22.1184MHZ, 11.0592MHZ.

;测试方法: 上位机发送数据,MCU收到数据后原样返回.

;串口固定设置: 1位起始位, 8位数据位, 1位停止位.

;*********************************************

STACK_POIRTER EQU     0D0H              ;堆栈开始地址

;UART3_BitTime   EQU   9216              ; 1200bps @ 11.0592MHz
                                         ;UART3_BitTime = (MAIN_Fosc / Baudrate)
;UART3_BitTime   EQU   4608              ; 2400bps @ 11.0592MHz
;UART3_BitTime   EQU   2304              ; 4800bps @ 11.0592MHz
;UART3_BitTime   EQU   1152              ; 9600bps @ 11.0592MHz
;UART3_BitTime   EQU    576              ;19200bps @ 11.0592MHz
;UART3_BitTime   EQU    288              ;38400bps @ 11.0592MHz

;UART3_BitTime   EQU   15360             ; 1200bps @ 18.432MHz
;UART3_BitTime   EQU   7680              ; 2400bps @ 18.432MHz
;UART3_BitTime   EQU   3840              ; 4800bps @ 18.432MHz
;UART3_BitTime   EQU   1920              ; 9600bps @ 18.432MHz
;UART3_BitTime   EQU   960               ;19200bps @ 18.432MHz
;UART3_BitTime   EQU   480               ;38400bps @ 18.432MHz
;UART3_BitTime   EQU   320               ;57600bps @ 18.432MHz

;UART3_BitTime   EQU   18432             ; 1200bps @ 22.1184MHz
;UART3_BitTime   EQU   9216              ; 2400bps @ 22.1184MHz
;UART3_BitTime   EQU   4608              ; 4800bps @ 22.1184MHz
;UART3_BitTime   EQU   2304              ; 9600bps @ 22.1184MHz
;UART3_BitTime   EQU   1152              ;19200bps @ 22.1184MHz
;UART3_BitTime   EQU   576               ;38400bps @ 22.1184MHz
UART3_BitTime    EQU   384               ;57600bps @ 22.1184MHz
;UART3_BitTime   EQU   27648             ; 1200bps @ 33.1776MHz
```

STC MCU Limited

```
;UART3_BitTime  EQU     13824           ; 2400bps @ 33.1776MHz
;UART3_BitTime  EQU     6912            ; 4800bps @ 33.1776MHz
;UART3_BitTime  EQU     3456            ; 9600bps @ 33.1776MHz
;UART3_BitTime  EQU     1728            ;19200bps @ 33.1776MHz
;UART3_BitTime  EQU     864             ;38400bps @ 33.1776MHz
;UART3_BitTime  EQU     576             ;57600bps @ 33.1776MHz
;UART3_BitTime  EQU     288             ;115200bps @ 33.1776MHz

PCA_P12_P11_P10_P37     EQU     (0 SHL 4)
PCA_P34_P35_P36_P37     EQU     (1 SHL 4)
PCA_P24_P25_P26_P27     EQU     (2 SHL 4)
PCA_Mode_Capture        EQU     0
PCA_Mode_SoftTimer      EQU     048H
PCA_Clock_1T            EQU     (4 SHL 1)
PCA_Clock_2T            EQU     (1 SHL 1)
PCA_Clock_4T            EQU     (5 SHL 1)
PCA_Clock_6T            EQU     (6 SHL 1)
PCA_Clock_8T            EQU     (7 SHL 1)
PCA_Clock_12T           EQU     (0 SHL 1)
PCA_Clock_ECI           EQU     (3 SHL 1)
PCA_Rise_Active         EQU     (1 SHL 5)
PCA_Fall_Active         EQU     (1 SHL 4)

ENABLE                  EQU     1

AUXR1           DATA    0xA2
CCON            DATA    0xD8
CMOD            DATA    0xD9
CCAPM0          DATA    0xDA            ; PCA模块0的工作模式寄存器。
CCAPM1          DATA    0xDB            ; PCA模块1的工作模式寄存器。
CCAPM2          DATA    0xDC            ; PCA模块2的工作模式寄存器。

CL              DATA    0xE9
CCAP0L          DATA    0xEA            ; PCA模块0的捕捉/比较寄存器低8位。
CCAP1L          DATA    0xEB            ; PCA模块1的捕捉/比较寄存器低8位。
CCAP2L          DATA    0xEC            ; PCA模块2的捕捉/比较寄存器低8位。

CH              DATA    0xF9
CCAP0H          DATA    0xFA            ; PCA模块0的捕捉/比较寄存器高8位。
CCAP1H          DATA    0xFB            ; PCA模块1的捕捉/比较寄存器高8位。
CCAP2H          DATA    0xFC            ; PCA模块2的捕捉/比较寄存器高8位。

CCF0    BIT     CCON.0                  ; PCA 模块0中断标志，由硬件置位，必须由软件清0。
CCF1    BIT     CCON.1                  ; PCA 模块1中断标志，由硬件置位，必须由软件清0。
CCF2    BIT     CCON.2                  ; PCA 模块2中断标志，由硬件置位，必须由软件清0。
CR      BIT     CCON.6                  ; 1: 允许PCA计数器计数，0: 禁止计数。
CF      BIT     CCON.7                  ; PCA计数器溢出（CH，CL由FFFFH变为0000H）标志。
                                        ;PCA计数器溢出后由硬件置位，必须由软件清0。
PPCA    BIT     IP.7                    ; PCA 中断 优先级设定位
```

STC MCU Limited

```
;====================== 模拟串口相关 ============================
P_RX3           BIT     P2.5            ; 定义模拟串口接收IO
P_TX3           BIT     P2.6            ; 定义模拟串口发送IO

Rx3_Ring        BIT     20H.0           ; 正在接收标志, 低层程序使用, 用户程序不可见
Tx3_TingBIT     20H.1                   ; 正在发送标志, 用户置1请求发送, 底层发送完成清0
RX3_End         BIT     20H.2           ; 接收到一个字节, 用户查询 并清0

Tx3_DAT         DATA    30H             ; 发送移位变量, 用户不可见
Rx3_DAT         DATA    31H             ; 接收移位变量, 用户不可见
Tx3_BitCnt      DATA    32H             ; 发送数据的位计数器, 用户不可见
Rx3_BitCnt      DATA    33H             ; 接收数据的位计数器, 用户不可见
Rx3_BUF         DATA    34H             ; 接收到的字节, 用户读取
Tx3_BUF         DATA    35H             ; 要发送的字节, 用户写入
;=================================================================
Tx3_read DATA   36H                     ; 发送读指针
Rx3_write       DATA    37H             ; 接收写指针

RX_Lenth        EQU     16              ; 接收长度
buf3            EQU     40H             ; 40H ~ 4FH       接收缓冲
;*****************************************************************
;*****************************************************************
        ORG     00H                     ;reset
        LJMP    F_Main

        ORG     3BH                     ;7  PCA interrupt
        LJMP    F_PCA_Interrupt

;******************** 主程序 **************************/
F_Main:

        MOV     SP, #STACK_POIRTER
        MOV     PSW, #0
        USING   0                       ;选择第0组R0~R7

;================= 用户初始化程序 ====================================
        LCALL   F_PCA_Init              ;PCA初始化
        SETB    EA

        MOV     Tx3_read,  #0
        MOV     Rx3_write, #0
        CLR     Tx3_Ting
        CLR     RX3_End
        CLR     Rx3_Ring
        MOV     Tx3_BitCnt, #0

;================== 主循环 ==================================
L_MainLoop:
        JNB             RX3_End, L_QuitRx3      ; 检测是否收到一个字节
        CLR             RX3_End                                 ; 清除标志
```

STC MCU Limited

```
        MOV     A, #buf3
        ADD     A, Rx3_write
        MOV     R0, A
        MOV     @R0, Rx3_BUF                ; 写入缓冲
        INC     Rx3_write                   ; 指向下一个位置
        MOV     A, Rx3_write
        CLR     C
        SUBB    A, #RX_Lenth                ; 溢出检测
        JC      L_QuitRx3
        MOV     Rx3_write, #0
L_QuitRx3:

        JB      Tx3_Ting, L_QuitTx3         ;检测是否发送空闲
        MOV     A, Tx3_read
        XRL     A, Rx3_write
        JZ      L_QuitTx3                   ;检测是否收到过字符

        MOV     A, #buf3
        ADD     A, Tx3_read
        MOV     R0, A
        MOV     Tx3_BUF, @R0                ; 从缓冲读一个字符发送
        SETB    Tx3_Ting                    ; 设置发送标志
        INC     Tx3_read                    ; 指向下一个字符位置
        MOV     A, Tx3_read
        CLR     C
        SUBB    A, #RX_Lenth
        JC      L_QuitTx3                   ; 溢出检测
        MOV     Tx3_read, #0
L_QuitTx3:

        SJMP    L_MainLoop

;=================== 主程序结束 ==================================
;================================================================================
; 函数: F_PCA_Init
; 描述: PCA初始化程序.
; 参数: none
; 返回: none.
; 版本: V1.0, 2013-11-22
;================================================================================
F_PCA_Init:
        CLR             CR
        MOV             CCAPM0, #(PCA_Mode_Capture OR PCA_Fall_Active OR ENABLE)
                                            ; 16位下降沿捕捉中断模式
        MOV             CCAPM1, #(PCA_Mode_SoftTimer OR ENABLE)
                                            ; 16位软件定时器, 中断模式
        MOV             CCAP1L, #LOW UART3_BitTime
                                            ; 将影射寄存器写入捕获寄存器，先写CCAP0L
        MOV             CCAP1H, #HIGH UART3_BitTime         ; 后写CCAP0H
```

STC MCU Limited

```
        MOV     CH, #0
        MOV     CL, #0
        MOV     A, AUXR1
        ANL     A, # NOT(3 SHL 4)
        ORL     A, #PCA_P24_P25_P26_P27         ;切换IO口
        MOV     AUXR1, A
        ANL     A, #NOT(7 SHL 1)
        ORL     A, #PCA_Clock_1T                ;选择时钟源
        MOV     CMOD, A
        SETB    PPCA                            ; 高优先级中断
        SETB    CR                              ; 运行PCA定时器
        RET
;=========================================================================
;=========================================================================
; 函数: F_PCA_Interrupt
; 描述: PCA中断处理程序.
; 参数: None
; 返回: none.
; 版本: V1.0, 2012-11-22
;=========================================================================
F_PCA_Interrupt:
        PUSH    PSW
        PUSH    ACC
        ;=============== PCA模块0中断 ====================
        JNB     CCF0, L_QuitPCA0                ; PCA模块0中断
        CLR     CCF0                            ; 清PCA模块0中断标志

        JNB     Rx3_Ring, L_Rx3_Start           ; 已收到起始位
        DJNZ    Rx3_BitCnt, L_RxBit             ; 接收完一帧数据

        CLR     Rx3_Ring                        ; 停止接收
        MOV     Rx3_BUF, Rx3_DAT                ; 存储数据到缓冲区
        SETB    RX3_End                                 ;
        MOV     CCAPM0, #(PCA_Mode_Capture OR PCA_Fall_Active OR ENABLE)
                                                ; 16位下降沿捕捉中断模式
        SJMP    L_QuitPCA0

L_RxBit:
        MOV     A, Rx3_DAT                      ; 把接收的单b数据 暂存到 RxShiftReg(接收缓冲)
        MOV     C, P_RX3
        RRC     A
        MOV     Rx3_DAT, A
        MOV     A, CCAP0L                       ;
        ADD     A, #LOW UART3_BitTime           ; 数据位时间
        MOV     CCAP0L, A                       ; 将影射寄存器写入捕获寄存器，先写CCAP0L
        MOV     A, CCAP0H                       ; 数据位时间
        ADDC    A, #HIGH UART3_BitTime  ;
        MOV     CCAP0H, A                       ; 后写CCAP0H
        SJMP    L_QuitPCA0
```

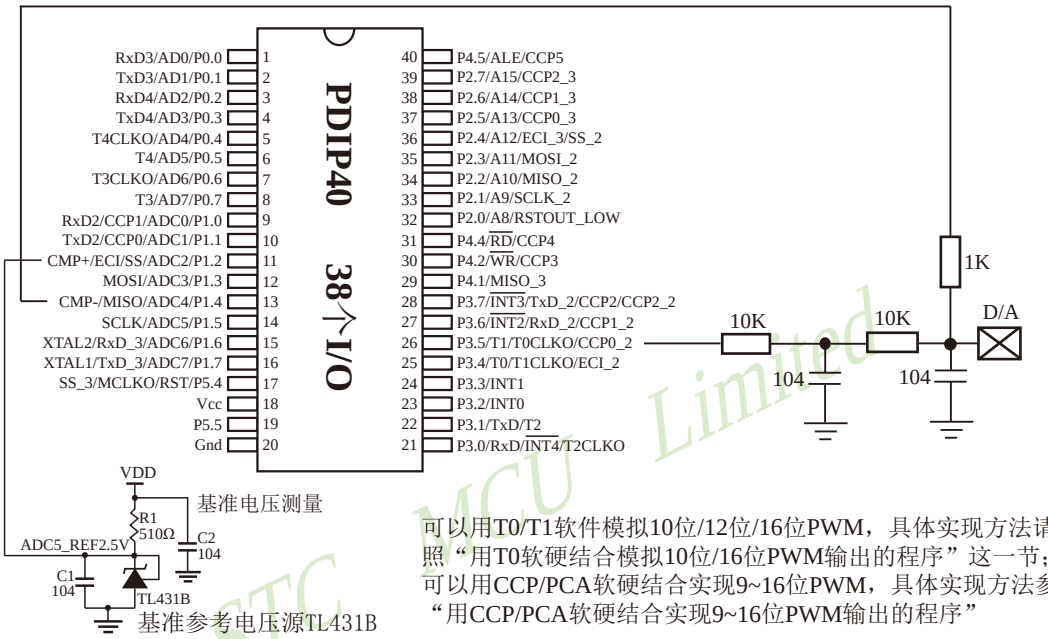
STC MCU Limited

```
L_Rx3_Start:
        MOV     CCAPM0, #(PCA_Mode_SoftTimer OR ENABLE)         ; 16位软件定时中断模式
        MOV     A, CCAP0L                                       ; 数据位时间
        ADD     A, #LOW (UART3_BitTime / 2 + UART3_BitTime)     ;
        MOV     CCAP0L, A                       ; 将影射寄存器写入捕获寄存器，先写CCAP0L
        MOV     A, CCAP0H                       ; 数据位时间
        ADDC    A, #HIGH (UART3_BitTime / 2 + UART3_BitTime)    ;
        MOV     CCAP0H, A                       ; 后写CCAP0H
        SETB    Rx3_Ring                        ; 标志已收到起始位
        MOV     Rx3_BitCnt, #9                  ; 初始化接收的数据位数(8个数据位+1个停止位)
L_QuitPCA0:

        ;================ PCA模块1中断 ====================
        JNB     CCF1, L_QuitPCA1                ; PCA模块1中断, 16位软件定时中断模式
        CLR     CCF1                            ; 清PCA模块1中断标志
        MOV     A, CCAP1L                       ;
        ADD     A, #LOW UART3_BitTime           ; 数据位时间
        MOV     CCAP1L, A                       ; 将影射寄存器写入捕获寄存器，先写CCAP0L
        MOV     A, CCAP1H                       ;
        ADDC    A, #HIGH UART3_BitTime          ; 数据位时间
        MOV     CCAP1H, A                       ; 后写CCAP0H

        JNB     Tx3_Ting, L_QuitPCA1            ; 不发送, 退出
        MOV     A, Tx3_BitCnt
        JNZ     L_TxData                        ; 发送计数器为0 表明单字节发送还没开始
        CLR     P_TX3                           ; 发送开始位
        MOV     Tx3_DAT, Tx3_BUF                ; 把缓冲的数据放到发送的buff
        MOV     Tx3_BitCnt, #9                  ; 发送数据位数 (8数据位+1停止位)
        SJMP    L_QuitPCA1
L_TxData:                                       ; 发送计数器为非0 正在发送数据
        DJNZ    Tx3_BitCnt, L_TxBit             ; 发送计数器减为0 表明单字节发送结束
        SETB    P_TX3                           ; 送停止位数据
        CLR     Tx3_Ting                        ; 发送停止
        SJMP    L_QuitPCA1
L_TxBit:
        MOV     A, Tx3_DAT                      ; 把最低位送到 CY(益处标志位)
        RRC     A
        MOV     P_TX3, C                        ; 发送一个bit数据
        MOV     Tx3_DAT, A
L_QuitPCA1:

        POP     ACC
        POP     PSW

        RETI

        END
```

STC MCU Limited

11.15 比利用CCP/PCA模块实现8~16位DAC的参考线路图

CCP：是英文单词的缩写
Capture(捕获),Compare(比较),PWM(脉宽调制)



可以用T0/T1软件模拟10位/12位/16位PWM，具体实现方法请参照“用T0软硬结合模拟10位/16位PWM输出的程序”这一节；还可以用CCP/PCA软硬结合实现9~16位PWM，具体实现方法参照“用CCP/PCA软硬结合实现9~16位PWM输出的程序”

如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

利用CCP/PCA模块的高速脉冲输出功能实现9~16位PWM来实现9~16位DAC，或用本身的硬件8位PWM来实现8位DAC，单片机本身也有10位ADC。

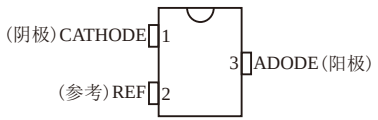
提示：

（1）PWM频率越高，输出波形越平滑。

（2）如果工作电压为5V，需输出1V电压，则设置高电平为1/5，低电平为4/5，则PWM输出电压就为1V。

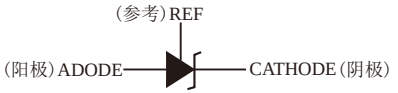
（3）如果要输出高精准电压，建议用A/D检测输出的电压值，然后根据A/D检测的电压值逐步调整到所需要的电压。

基准参考电压源TL431B



SOT23-3封装，RMB￥0.15~0.3

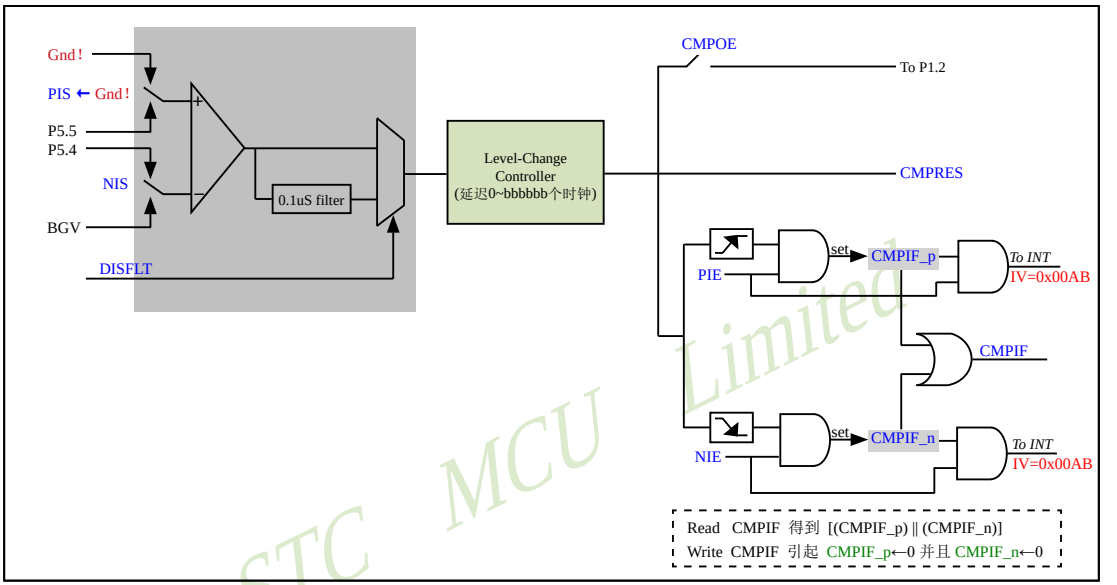
基准参考电压源TL431B的符号



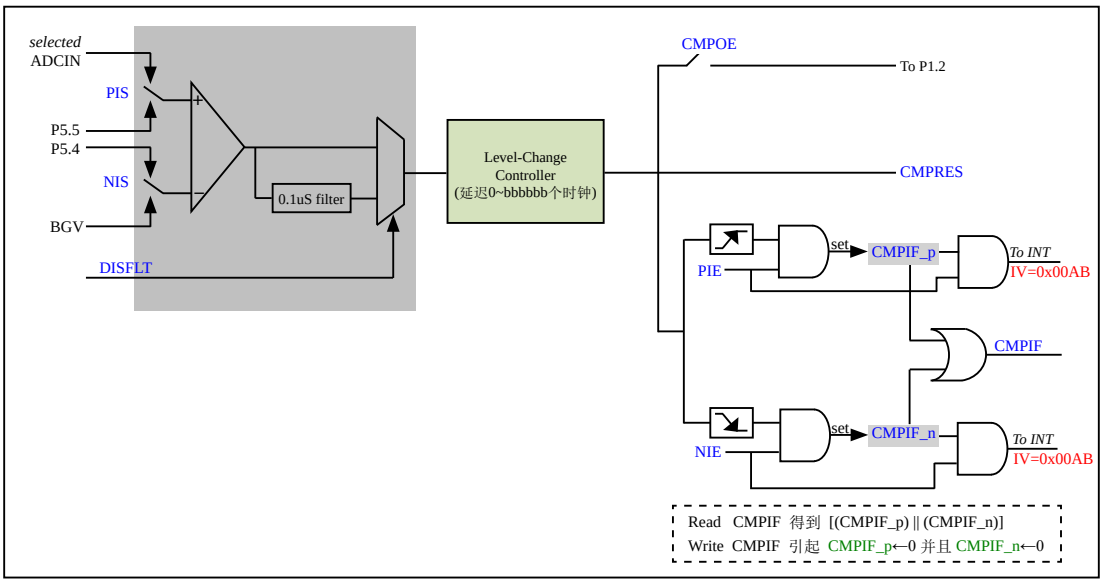
如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

第12章 STC15W系列的比较器

STC15W系列单片机(如STC15W408AS系列、STC15W201S系列、STC15W408S系列、STC15W1K16S系列及STC15W4K60S4系列)内置比较器功能。其中STC15W201S系列、STC15W408S系列及STC15W1K16S系列的比较器内部规划如下图所示:



其中，有ADC的单片机STC15W408AS系列及STC15W4K60S4系列的比较器内部规划如下图所示：



STC MCU Limited

STC15W系列与比较器相关的特殊功能寄存器(STC15W SFRs associated with comparator)

符号	描述	地址	位地址及其符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	比较器控制寄存器1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,1001

1. 比较器控制寄存器1：CMPCR1

比较器控制寄存器1的格式如下：

CMPCR1：比较器控制寄存器1

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	name	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPEN：比较器模块使能位

CMPEN=1，使能比较器模块；

CMPEN=0，禁用比较器模块, 比较器的电源关闭。

CMPIF：比较器中断标志位(Interrupt Flag)

在 CMPEN为1的情况下：

当比较器的比较结果由LOW变成HIGH时, 若是PIE被设置成1, 那么内建的某一个叫做CMPIF_p的寄存器会被设置成1；

当比较器的比较结果由HIGH变成LOW时, 若是NIE被设置成1, 那么内建的某一个叫做CMPIF_n的寄存器会被设置成1；

当CPU 去读取 CMPIF 的数值时, 会读到 (CMPIF_p || CMPIF_n);

当CPU对CMPIF写0后, CMPIF_p 以及 CMPIF_n 都会被清除为0 .

而中断产生的条件是 [(EA==1) && (((PIE==1)&&(CMPIF_p==1)) || ((NIE==1)&&(CMPIF_n==1)))]

CPU接受中断后, 并不会自动清除此CMPIF标志, 用户必须用软件写”0”去清除它。

PIE：比较器上升沿中断使能位(Pos-edge Interrupt Enabling)

PIE = 1，使能比较器由LOW变HIGH的事件 设定CMPIF_p/产生中断；

PIE = 0，禁用比较器由LOW变HIGH的事件 设定CMPIF_p/产生中断。

NIE：比较器下降沿中断使能位 (Neg-edge Interrupt Enabling)

NIE = 1，使能比较器由HIGH变LOW的事件 设定CMPIF_n/产生中断；

NIE = 0，禁用比较器由HIGH变LOW的事件 设定CMPIF_n/产生中断。

PIS：比较器正极选择位

PIS = 1，选择 ADCIS[2:0] 所选择到的 ADCIN 做为比较器的正极输入源；

PIS = 0，选择外部P5.5为比较器的正极输入源。

STC MCU Limited

NIS：比较器负极选择位

NIS = 1，选择外部管脚P5.4为比较器的负极输入源；

NIS = 0，选择内部BandGap电压BGV为比较器的负极输入源。

CMPOE：比较结果输出控制位

CMPOE = 1，使能比较器的比较结果输出到P1.2；

CMPOE = 0，禁止比较器的比较结果输出。

CMPRES：比较器比较结果 (Comparator Result)标志位

CMPRES = 1，CMP+的电平高于CMP-的电平(或内部BandGap参考电压的电平)；

CMPRES = 0，CMP+的电平低于CMP-的电平(或内部BandGap参考电压的电平)。

此 bit 是一个”只读(read-only)”的bit ; 软件对它做写入的动作没有任何意义。软件所读到的结果是“经过ENLCCTL控制后的结果”, 而非Analog比较器的直接输出结果.

2. 比较器控制寄存器2：CMPCR2

比较器控制寄存器2的格式如下：

CMPCR2 : 比较器控制寄存器2

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	name	INVCMPO	DISFLT	LCDTY[5:0]					

INVCMPO：比较器输出取反控制位 (Inverse Comparator Output)

INVCMPO = 1，比较器取反后再输出到P1.2；

INVCMPO = 0，比较器正常输出。

比较器的输出, 采用“经过ENLCCTL控制后的结果”, 而非Analog比较器的直接输出结果.

DISFLT：去除比较器输出的 0.1uS Filter

DISFLT = 1，关掉比较器输出的0.1uS Filter (可以让比较器速度有少许提升)；

DISFLT = 0，比较器的输出有 0.1uS 的 Filter。

LCDTY[5:0]：比较器输出端 Level-Change control的 filter 长度(Duty)选择

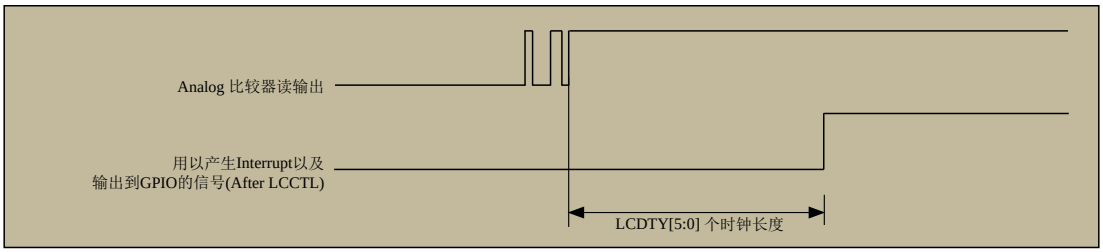
bbbbbb：=

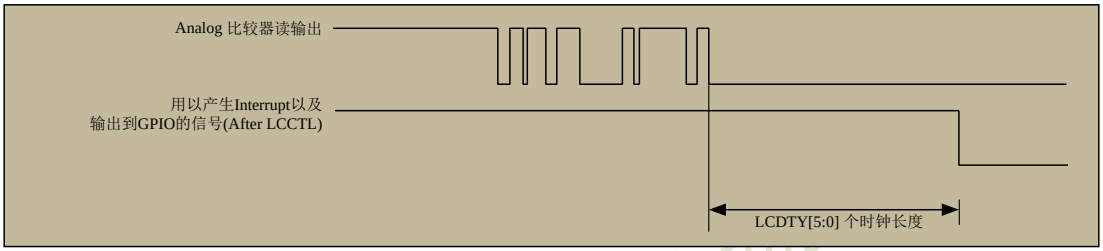
当比较器由LOW变HIGH, 必须侦测到该后来的HIGH持续至少bbbbbb个时钟, 此芯片线路才认定比较器的输出是由LOW转成HIGH; 如果在bbbbbb个时钟内, Analog比较器的输出又回复到LOW, 此芯片线路认为甚么都没发生, 视同比较器的输出一直维持在LOW；

当比较器由HIGH变LOW, 必须侦测到该后来的LOW持续至少bbbbbb个时钟, 此芯片线路才认定比较器的输出是由HIGH转成LOW; 如果在bbbbbb个时钟内, Analog比较器的输出又回复到HIGH, 此芯片线路认为甚么都没发生, 视同比较器的输出一直维持在HIGH.

若是设定成 000000, 代表没有 Level-Change Control.

STC MCU Limited





STC MCU Limited

12.1 比较器中断方式程序举例(C及汇编)

1.C语言程序

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 比较器中断方式举例-------------------------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8052芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1  =   0xE6;                   //比较器控制寄存器1
#define CMPEN       0x80                    //CMPCR1.7 : 比较器模块使能位
```

```
#define   CMPIF          0x40          //CMPCR1.6 : 比较器中断标志位
#define   PIE            0x20          //CMPCR1.5 : 比较器上升沿中断使能位
#define   NIE            0x10          //CMPCR1.4 : 比较器下降沿中断使能位
#define   PIS            0x08          //CMPCR1.3 : 比较器正极选择位
#define   NIS            0x04          //CMPCR1.2 : 比较器负极选择位
#define   CMPOE          0x02          //CMPCR1.1 : 比较结果输出控制位
#define   CMPRES         0x01          //CMPCR1.0 : 比较器比较结果标志位

sfr       CMPCR2    =    0xE7;         //比较器控制寄存器2
#define   INVCMPO        0x80          //CMPCR2.7 : 比较结果反向输出控制位
#define   DISFLT         0x40          //CMPCR2.6 : 比较器输出端虑波使能控制位
#define   LCDTY          0x3F          //CMPCR2.[5:0] : 比较器输出的区抖时间控制

sbit      LED       =    P1^1;         //测试脚

void cmp_isr() interrupt 21 using 1    //比较器中断向量入口
{
        CMPCR1 &= ~CMPIF;              //清除完成标志
        LED = !!(CMPCR1 & CMPRES);     //将比较器结果CMPRES输出到测试口显示
}

void main()
{
        CMPCR1 = 0;                    //初始化比较器控制寄存器1
        CMPCR2 = 0;                    //初始化比较器控制寄存器2

        CMPCR1 &= ~PIS;                //选择外部管脚P5.5（CMP+）为比较器的正极输入源
//      CMPCR1 |= PIS;                 //选择ADCIS[2:0]所选的ADCIN为比较器的正极输入源

        CMPCR1 &= ~NIS;                //选择内部BandGap电压BGV为比较器的负极输入源
//      CMPCR1 |= NIS;                 //选择外部管脚P5.4（CMP-）为比较器的负极输入源

        CMPCR1 &= ~CMPOE;              //禁用比较器的比较结果输出
//      CMPCR1 |= CMPOE;               //使能比较器的比较结果输出到P1.2

        CMPCR2 &= ~INVCMPO;            //比较器的比较结果正常输出到P1.2
//      CMPCR2 |= INVCMPO;             //比较器的比较结果取反后输出到P1.2

        CMPCR2 &= ~DISFLT;             //不禁用(使能)比较器输出端的0.1uS虑波电路
//      CMPCR2 |= DISFLT;              //禁用比较器输出端的0.1uS虑波电路

        CMPCR2 &= ~LCDTY;              //比较器结果不去抖动,直接输出
//      CMPCR2 |= (DISFLT & 0x10);     //比较器结果在经过16个时钟后再输出
```

STC MCU Limited

```
        CMPCR1 |= PIE;                  /使能比较器的上升沿中断
//      CMPCR1 |= NIE;                  //使能比较器的下降沿中断

        CMPCR1 |= CMPEN;                //使能比较器

        EA = 1;

        while (1);
}
```

2.汇编程序

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 比较器中断方式举例-------------------------------------------------*/
/* --- 研发顾问QQ：800003751---------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8052芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz

CMPCR1      DATA    0E6H        ;比较器控制寄存器1
CMPEN       EQU     080H        ;CMPCR1.7 : 比较器模块使能位
CMPIF       EQU     040H        ;CMPCR1.6 : 比较器中断标志位
PIE         EQU     020H        ;CMPCR1.5 : 比较器上升沿中断使能位
NIE         EQU     010H        ;CMPCR1.4 : 比较器下降沿中断使能位
PIS         EQU     008H        ;CMPCR1.3 : 比较器正极选择位
NIS         EQU     004H        ;CMPCR1.2 : 比较器负极选择位
CMPOE       EQU     002H        ;CMPCR1.1 : 比较结果输出控制位
CMPRES      EQU     001H        ;CMPCR1.0 : 比较器比较结果标志位

CMPCR2      DATA    0E7H        ;比较器控制寄存器2
INVCMPO     EQU     080H        ;CMPCR2.7 : 比较结果反向输出控制位
DISFLT      EQU     040H        ;CMPCR2.6 : 比较器输出端虑波使能控制位
LCDTY       EQU     03FH        ;CMPCR2.[5:0] : 比较器输出的区抖时间控制

LED         BIT     P1.1        ;测试脚
```

STC MCU Limited

```
;-----------------------------------------
        ORG     0000H
        LJMP    MAIN

        ORG     00ABH
        LJMP    CMP_ISR                 ;比较器中断向量入口

;-----------------------------------------
        ORG     0100H
MAIN:
        MOV     CMPCR1,     #0          ;初始化比较器控制寄存器
        MOV     CMPCR2,     #0          ;初始化比较器控制寄存器

        ANL     CMPCR1,     #NOT PIS
                                        ;选择外部管脚P5.5（CMP+）为比较器的正极输入源
//      ORL     CMPCR1,     #PIS        ;选择ADCIS[2:0]所选的ADCIN为比较器的正极输入源

        ANL     CMPCR1,     #NOT NIS
                                        ;选择内部BandGap电压BGV为比较器的负极输入源
//      ORL     CMPCR1,     #NIS        ;选择外部管脚P5.4（CMP-）为比较器的负极输入源

        ANL     CMPCR1,     #NOT CMPOE
                                        ;禁用比较器的比较结果输出
//      ORL     CMPCR1,     #CMPOE
                                        ;使能比较器的比较结果输出到P1.2

        ANL     CMPCR2,     #NOT INVCMPO
                                        ;比较器的比较结果正常输出到P1.2
//      ORL     CMPCR2,     #INVCMPO
                                        ;比较器的比较结果取反后输出到P1.2

        ANL     CMPCR2,     #NOT DISFLT
                                        ;不禁用(使能)比较器输出端的0.1uS虑波电路
//      ORL     CMPCR2,     #DISFLT
                                        ;禁用比较器输出端的0.1uS虑波电路

        ANL     CMPCR2,     #NOT LCDTY
                                        ;比较器结果不去抖动,直接输出
//      ORL     CMPCR2,     #(DISFLT AND 0x10)
                                        ;比较器结果在经过16个时钟后再输出

        ORL     CMPCR1,     #PIE        ;使能比较器的上升沿中断
//      ORL     CMPCR1,     #NIE        ;使能比较器的下降沿中断
```

STC MCU Limited

```
        ORL     CMPCR1,         #CMPEN          ;使能比较器

        SETB    EA

        SJMP    $

;----------------------------------------
CMP_ISR:
        PUSH    PSW
        PUSH    ACC

        ANL     CMPCR1,         #NOT CMPIF      ;清除完成标志
        MOV     A,      CMPCR1
        MOV     C,      ACC.0                   ;将比较器结果CMPRES输出到测试口显示
        MOV     LED,    C

        POP     ACC
        POP     PSW
        RETI
;----------------------------------------
        END
```

STC MCU Limited

12.2 比较器查询方式程序举例(C及汇编)

1.C语言程序

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 比较器查询方式举例------------------------------------------------------*/
/* --- 研发顾问QQ：800003751---------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8052芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1=     0xE6;     //比较器控制寄存器1
#define CMPEN       0x80      //CMPCR1.7 : 比较器模块使能位
#define CMPIF       0x40      //CMPCR1.6 : 比较器中断标志位
#define PIE         0x20      //CMPCR1.5 : 比较器上升沿中断使能位
#define NIE         0x10      //CMPCR1.4 : 比较器下降沿中断使能位
#define PIS         0x08      //CMPCR1.3 : 比较器正极选择位
#define NIS         0x04      //CMPCR1.2 : 比较器负极选择位
#define CMPOE       0x02      //CMPCR1.1 : 比较结果输出控制位
#define CMPRES      0x01      //CMPCR1.0 : 比较器比较结果标志位

sfr     CMPCR2  =   0xE7;     //比较器控制寄存器2
#define INVCMPO     0x80      //CMPCR2.7 : 比较结果反向输出控制位
#define DISFLT      0x40      //CMPCR2.6 : 比较器输出端0.1us虑波控制位
#define LCDTY       0x3F      //CMPCR2.[5:0] : 比较器输出的区抖时间控制

sbit    LED     =   P1^1;     //测试脚

void main()
{
    CMPCR1 = 0;               //初始化比较器控制寄存器1
    CMPCR2 = 0;               //初始化比较器控制寄存器2

    CMPCR1 &= ~PIS;           //选择外部管脚P5.5（CMP+）为比较器的正极输入源
//  CMPCR1 |= PIS;            //选择ADCIS[2:0]所选的ADCIN为比较器的正极输入源

    CMPCR1 &= ~NIS;           //选择内部BandGap电压BGV为比较器的负极输入源
//  CMPCR1 |= NIS;            //选择外部管脚P5.4（CMP-）为比较器的负极输入源
```

STC MCU Limited

```
        CMPCR1 &= ~CMPOE;               //禁用比较器的比较结果输出
//      CMPCR1 |= CMPOE;                //使能比较器的比较结果输出到P1.2

        CMPCR2 &= ~INVCMPO;             //比较器的比较结果正常输出到P1.2
//      CMPCR2 |= INVCMPO;              //比较器的比较结果取反后输出到P1.2

        CMPCR2 &= ~DISFLT;              //不禁用(使能)比较器输出端的0.1uS虑波电路
//      CMPCR2 |= DISFLT;               //禁用比较器输出端的0.1uS虑波电路

        CMPCR2 &= ~LCDTY;               //比较器结果不去抖动,直接输出
//      CMPCR2 |= (DISFLT & 0x10);      //比较器结果在经过16个时钟后再输出

        CMPCR1 |= CMPEN;                //使能比较器
        while (!(CMPCR1 & CMPIF));      //查询比较完成标志
        CMPCR1 &= ~CMPIF;               //清除完成标志
        LED = !!(CMPCR1 & CMPRES);      //将比较器结果CMPRES输出到测试口显示

        while (1);
}
```

2.汇编程序

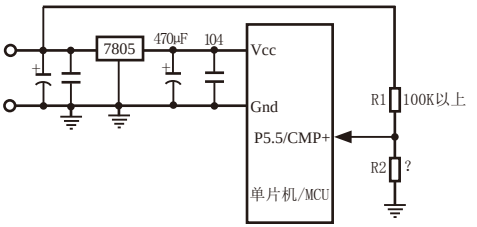
```
/*-------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 比较器查询方式举例------------------------------------------------*/
/* --- 研发顾问QQ：800003751---------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*-------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8052芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz

CMPCR1      DATA    0E6H    ;比较器控制寄存器1
CMPEN       EQU     080H    ;CMPCR1.7 : 比较器模块使能位
CMPIF       EQU     040H    ;CMPCR1.6 : 比较器中断标志位
PIE         EQU     020H    ;CMPCR1.5 : 比较器上升沿中断使能位
NIE         EQU     010H    ;CMPCR1.4 : 比较器下降沿中断使能位
PIS         EQU     008H    ;CMPCR1.3 : 比较器正极选择位
NIS         EQU     004H    ;CMPCR1.2 : 比较器负极选择位
CMPOE       EQU     002H    ;CMPCR1.1 : 比较结果输出控制位
CMPRES      EQU     001H    ;CMPCR1.0 : 比较器比较结果标志位
```

STC MCU Limited

```
CMPCR2          DATA    0E7H    ;比较器控制寄存器2
INVCMPO         EQU     080H    ;CMPCR2.7 : 比较结果反向输出控制位
DISFLT          EQU     040H    ;CMPCR2.6 : 比较器输出端虑波使能控制位
LCDTY           EQU     03FH    ;CMPCR2.[5:0] : 比较器输出的区抖时间控制

LED             BIT     P1.1    ;测试脚

;----------------------------------------
        ORG     0000H
        LJMP    MAIN
;----------------------------------------
        ORG     0100H
MAIN:
        MOV     CMPCR1,         #0          ;初始化比较器控制寄存器
        MOV     CMPCR2,         #0          ;初始化比较器控制寄存器

        ANL     CMPCR1,         #NOT PIS ;选择外部管脚P5.5（CMP+）为比较器的正极输入源
//      ORL     CMPCR1,         #PIS        ;选择ADCIS[2:0]所选的ADCIN为比较器的正极输入源

        ANL     CMPCR1,         #NOT NIS
                                            ;选择内部BandGap电压BGV为比较器的负极输入源
//      ORL     CMPCR1,         #NIS        ;选择外部管脚P5.4（CMP-）为比较器的负极输入源

        ANL     CMPCR1,         #NOT CMPOE      ;禁用比较器的比较结果输出
//      ORL     CMPCR1,         #CMPOE          ;使能比较器的比较结果输出到P1.2

        ANL     CMPCR2,         #NOT INVCMPO    ;比较器的比较结果正常输出到P1.2
//      ORL     CMPCR2,         #INVCMPO        ;比较器的比较结果取反后输出到P1.2

        ANL     CMPCR2,         #NOT DISFLT     ;不禁用(使能)比较器输出端的0.1uS虑波电路
//      ORL     CMPCR2,         #DISFLT         ;禁用比较器输出端的0.1uS虑波电路

        ANL     CMPCR2,         #NOT LCDTY      ;比较器结果不去抖动,直接输出
//      ORL     CMPCR2,         #(DISFLT AND 0x10)      ;比较器结果在经过16个时钟后再输出

        ORL     CMPCR1,         #CMPEN          ;使能比较器
WAIT:
        MOV     A,              CMPCR1          ;查询比较完成标志
        ANL     A,              #CMPIF
        JZ      WAIT
        ANL     CMPCR1,         #NOT CMPIF      ;清除完成标志
        MOV     A,              CMPCR1
        MOV     C,              ACC.0           ;将比较器结果CMPRES输出到测试口显示
        MOV     LED,            C

        SJMP    $

        END
```

STC MCU Limited

12.3 比较器作外部掉电检测的参考电路



上图中，电阻R1和R2对稳压块7805的前端电压进行分压，分压后的电压作为P5.5/CMP+的外部输入与内部BandGap参考电压(1.27V附近)进行比较。

一般当交流电在220V时，稳压块7805前端的直流电压是11V，但当交流电压降到160V时，稳压块7805前端的直流电压是8.5V。当稳压块7805前端的直流电压低于或等于8.5V时，该前端输入的直流电压被电阻R1和R2分压到CMP+端(比较器正极输入端)，CMP+端输入电压低于内部BandGap参考电压(1.27V附近)，此时可产生比较器中断，这样在掉电检测时就有充足的时间将数据保存到EEPROM中。当稳压块7805前端的直流电压高于8.5V时，该前端输入的直流电压被电阻R1和R2分压到CMP+端(比较器正极输入端)，CMP+端输入电压高于内部BandGap参考电压(1.27V附近)，此时CPU可继续正常工作。

内部BandGap参考电压约在1.27V附近，具体数值要通过读取内部BandGap电压在内部RAM区或ROM区所占用的地址的值获得。对于具有128字节RAM空间的单片机（如STC15F104W系列单片机），其内部BandGap参考电压值在RAM区占用的地址为06FH-070H，在ROM区占用的地址为程序空间最后第8字节和第9字节（如STC15F104W型号单片机具有4K程序空间，则其内部BandGap参考电压值在ROM区占用的地址为0FF7H-0FF8H），用户只需通过读取RAM区06FH-070H地址的值或ROM区0FF7H-0FF8H地址的值即可获得STC15F104W型号单片机的内部BandGap参考电压值(毫伏,高字节在前)。对于具有256及其以上字节RAM空间的单片机（如STC15F2K60S2系列单片机），其内部BandGap参考电压值在RAM区占用的地址为0EFH-0F0H，在ROM区占用的地址为程序空间最后第8字节和第9字节（如STC15F2K60S2型号单片机具有60K程序空间，则其内部BandGap参考电压值在ROM区占用的地址为EFF7H-EFF8H），用户只需通过读取RAM区0EFH-0F0H地址的值或ROM区EFF7H-EFF8H地址的值即可获得STC15F2K60S2型号单片机的内部BandGap参考电压值(毫伏,高字节在前)。

12.4 STC15W系列比较器作ADC的程序举例(C语言)

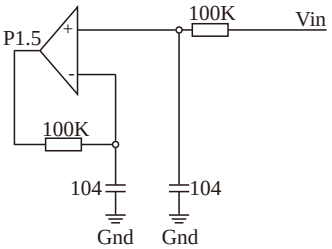
```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 比较器作ADC的程序举例----------------------------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8052芯片型号进行编译

//假定测试芯片的工作频率为22.1184MHz

//使用MCU自带的比较器进行ADC转换, 并经过模拟串口输出结果.

/*****************************
```

使用比较器做ADC, 原理图如下.
做ADC的原理是基于电荷平衡的计数式ADC.
电压从Vin输入, 通过100K+104滤波, 进入比较器的P5.5正输入端, 经过比较器的比较, 将结果输出到P1.5再通过100K+104滤波后送比较器P5.4负输入端,跟输入电压平衡.
设置两个变量: 计数周期(量程)adc_duty 和 比较结果高电平的计数值 adc, adc严格比例于输入电压.
ADC的基准就是P1.5的高电平. 如果高电平准确,比较器的放大倍数足够大,则ADC结果会很准确.
当比较结果为高电平,则P1.5输出1, 并且adc+1.
当比较结果为低电平,则P1.5输出0.
每一次比较都判断计数周期是否完成,完成则adc里的值就是ADC结果.
电荷平衡计数式ADC的性能类似数字万用表用的双积分ADC, 当计数周期为20ms的倍数时,具有很强的抗工频干扰能力,很好的线性和精度.
原理可以参考ADD3501(3 1/2位数字万用表)或ADD3701(3 3/4位数字万用表), 也可以参考AD7740 VFC电路.

例: 比较一次的时间间隔为10us, 量程为10000, 则做1次ADC的时间为100ms. 比较器的响应时间越短, 则完成ADC就越快.

由于要求每次比较时间间隔都要相等,所以用C编程最好在定时器中断里进行, 定时器设置为自动重装, 高优先级中断, 其它中断均低优先级.
用汇编的话, 保证比较输出电平处理的时间要相等.

STC MCU Limited



**********/

```
#include  "reg51.h"
#include  "intrins.h"
typedef   unsigned char     u8;
typedef   unsigned int      u16;
typedef   unsigned long     u32;

#define   MAIN_Fosc         22118400L                 //定义主时钟


sfr       P1M1    =         0x91;     //P1M1.n,P1M0.n = 00--->Standard, 01--->push-pull,实际上1T的都一样
sfr       P1M0    =         0x92;     //                =10--->pure input,11--->open drain
sfr       AUXR    =         0x8E;

sfr       CMPCR1 =          0xE6;                     //比较器控制寄存器1
#define   CMPEN             0x80                      //CMPCR1.7 : 比较器模块使能位
#define   CMPIF             0x40                      //CMPCR1.6 : 比较器中断标志位
#define   PIE               0x20                      //CMPCR1.5 : 比较器上升沿中断使能位
#define   NIE               0x10                      //CMPCR1.4 : 比较器下降沿中断使能位
#define   PIS               0x08                      //CMPCR1.3 : 比较器正极选择位
#define   NIS               0x04                      //CMPCR1.2 : 比较器负极选择位
#define   CMPOE             0x02                      //CMPCR1.1 : 比较结果输出控制位
#define   CMPRES            0x01                      //CMPCR1.0 : 比较器比较结果标志位

sfr       CMPCR2    =       0xE7;                     //比较器控制寄存器2
#define   INVCMPO           0x80                      //CMPCR2.7 : 比较结果反向输出控制位
#define   DISFLT            0x40                      //CMPCR2.6 : 比较器输出端0.1us虑波控制位
#define   LCDTY             0x3F                      //CMPCR2.[5:0] : 比较器输出的区抖时间控制

sbit      LED     =         P1^1;                     //测试脚

#define   ADC_SCALE         50000                     //ADC满量程, 根据需要设置
sbit      P_ADC   =         P1^5;                     //P1.2 比较器输出端
unsigned int        adc;                              //ADC中间值, 用户层不可见
```

STC MCU Limited

```
unsigned int    adc_duty;                   //ADC计数周期, 用户层不可见
unsigned int    adc_value;                  //ADC值, 用户层使用
bit             adc_ok;                     //ADC结束标志, 为1则adc_value的值可用.
                                            //此标志给用户层查询,并且清0
sbit    P_TXD   =       P3^1;               //定义模拟串口发送端,可以是任意IO

void    TxSend(u8 dat);
void    PrintString(unsigned char code *puts);

void main()
{
        u8      i;
        u8      tmp[5];

        CMPCR1 = 0;                         //初始化比较器控制寄存器1
        CMPCR2 = 0;                         //初始化比较器控制寄存器2

        CMPCR1 &= ~PIS;                     //选择外部管脚P5.5（CMP+）为比较器的正极输入源
//      CMPCR1 |= PIS;                      //选择ADCIS[2:0]所选的ADCIN为比较器的正极输入源

//      CMPCR1 &= ~NIS;                     //选择内部BandGap电压BGV为比较器的负极输入源
        CMPCR1 |= NIS;                      //选择外部管脚P5.4（CMP-）为比较器的负极输入源

        CMPCR1 &= ~CMPOE;                   //禁用比较器的比较结果输出
//      CMPCR1 |= CMPOE;                    //使能比较器的比较结果输出到P1.2

        CMPCR2 &= ~INVCMPO;                 //比较器的比较结果正常输出到P1.2
//      CMPCR2 |= INVCMPO;                  //比较器的比较结果取反后输出到P1.2

        CMPCR2 &= ~DISFLT;                  //使能比较器输出端的0.1uS虑波电路
//      CMPCR2 |= DISFLT;                   //禁用比较器输出端的0.1uS虑波电路

        CMPCR2 &= ~LCDTY;                   //比较器结果不去抖动,直接输出
//      CMPCR2 |= (DISFLT & 0x10);          //比较器结果在经过16个时钟后再输出

        CMPCR1 |= CMPEN;                    //使能比较器
//      while (!(CMPCR1 & CMPIF));          //查询比较完成标志
//      CMPCR1 &= ~CMPIF;                   //清除完成标志
//      LED = !!(CMPCR1 & CMPRES);          //将比较器结果CMPRES输出到测试口显示

        ET0 = 1;                            //允许中断
        PT0 = 1;                            //高优先级中断
        TMOD &= ~0x03;                      //工作模式,0: 16位自动重装, 1: 16位定时/计数,
                                            //2: 8位自动重装, 3: 16位自动重装, 不可屏蔽中断
```

STC MCU Limited

```
    AUXR |=  0x80;                          //1T

    TH0 = (u8)((-(MAIN_Fosc*10)/1000000) >> 8); //10us
    TL0 = (u8)(-(MAIN_Fosc*10)/1000000);

    P1M1 &= ~(1<<5);                        //P1.5设置为push pull输出
    P1M0 |=  (1<<5);

    adc_duty = ADC_SCALE;                   //周期计数赋初值
    adc = 0;
    TR0 = 1;                                //开始运行
    EA  = 1;

    PrintString("\r\n使用比较器做ADC例子\r\n");

    while (1)
    {
        if(adc_ok)
        {
            adc_ok = 0;                                 //清除ADC已结束标志
            PrintString("ADC = ");                      //转十进制
            tmp[0] = adc_value / 10000 + '0';
            tmp[1] = adc_value % 10000 / 1000 + '0';
            tmp[2] = adc_value % 1000 / 100 + '0';
            tmp[3] = adc_value % 100 / 10 + '0';
            tmp[4] = adc_value % 10 + '0';
            for(i=0; i<4; i++)                          //消无效0
            {
                if(tmp[i] != '0')       break;
                tmp[i] = ' ';
            }
            for(i=0; i<5; i++)    TxSend(tmp[i]);       //发串口
            PrintString("\r\n");
        }
    }
}

/********************** Timer0中断函数************************/
void timer0_int (void) interrupt 1
{
    if(CMPCR1 & CMPRES)                     //比较器输出高电平
    {
        P_ADC = 1;                          //P_ADC输出高电平, 给负输入端做反馈.
        adc ++;                             //ADC计数+1
    }
```

STC MCU Limited

```
        else    P_ADC = 0;                      //P_ADC输出低电平, 给负输入端做反馈.
        if(--adc_duty == 0)                     //ADC周期-1, 到0则ADC结束
        {
                adc_duty = ADC_SCALE;           //周期计数赋初值
                adc_value = adc;                //保存ADC值
                adc = 0;                        //清除ADC值
                adc_ok = 1;                     //标志ADC已结束
        }
}
//========================================================================
// 函数: void    BitTime(void)
// 描述: 位时间函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//========================================================================
void    BitTime(void)
{
        u16 i;
        i = ((MAIN_Fosc / 100) * 104) / 130000L - 1;            //根据主时钟来计算位时间
        while(--i);
}

//========================================================================
// 函数: void    TxSend(uchar dat)
// 描述: 模拟串口发送一个字节。9600，N，8，1
// 参数: dat: 要发送的数据字节.
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//========================================================================
void    TxSend(u8 dat)
{
        u8      i;
        EA = 0;
        P_TXD = 0;
        BitTime();
        for(i=0; i<8; i++)
        {
                if(dat & 1)             P_TXD = 1;
                else                    P_TXD = 0;
```

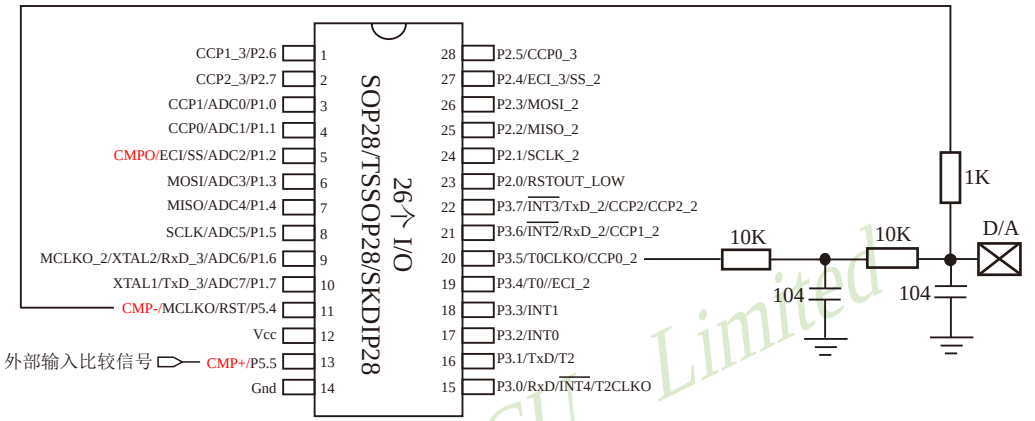
STC MCU Limited

```
                dat >>= 1;
                BitTime();
        }
        P_TXD = 1;
        EA = 1;
        BitTime();
        BitTime();
}
//=========================================================================
// 函数: void PrintString(unsigned char code *puts)
// 描述: 模拟串口发送一串字符串。9600，N，8，1
// 参数: *puts: 要发送的字符指针.
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//=========================================================================
void PrintString(unsigned char code *puts)
{
        for (; *puts != 0;     puts++)  TxSend(*puts);
}
```

STC MCU Limited

12.5 在比较器负端产生不同的电压由比较器正端进行比较

对于无ADC功能的单片机，如要进行电流检测或检测外部电池电压等，可以利用CCP/PWM内部产生一个电压输入到比较器负端(CPM-)，然后由比较器的正端(CMP+)将其与外部电压进行比较，从而达到检测外部电压的目的。具体实现的参考电路图如下：



PWM功能可以利用T0/T1软件模拟10位/12位/16位PWM来实现，具体实现方法请参照“用T0软硬结合模拟10位/16位PWM输出的程序”这一节；还可以利用CCP/PCA软硬结合实现9~16位PWM来实现，具体实现方法参照“用CCP/PCA软硬结合实现9~16位PWM输出的程序”。

STC MCU Limited

12.6 现供货的STC15W201S系列A版本的比较器下降沿中断不响应——将在STC15W201S系列B版本中修正

STC15W201S系列的A版本芯片正大批量现货供应中，当仅允许该版本的比较器下降沿中断时，该比较器的下降沿中断暂不能使用。但是，该版本的比较器下降沿中断不是绝对不能使用，用户可以通过以下两种办法解决这一问题：一、STC15W201S系列A版本的比较器上升沿中断是可正常使用的，且当用户将其比较器上升沿中断和下降沿中断都允许后，该比较器上升沿中断和下降沿中断都可以正常使用。由于比较器比较结果标志位CMPRES(CMPCR1.0)是正确的，因此在比较器中断服务程序中查询比较器比较结果标志位CMPRES(CMPCR1.0)的值可判断单片机进入的是比较器上升沿中断还是比较器下降沿中断，如果CMPRES/CMPCR1.0=1，即CMP+的电平高于CMP-的电平(或内部BandGap参考电压的电平)，则表示单片机进入的是比较器上升沿中断；反之，如果CMPRES(CMPCR1.0)=0，即CMP+的电平低于CMP-的电平(或内部BandGap参考电压的电平)，则表示单片机进入的是比较器下降沿中断，此时比较器下降沿中断是可正常使用的，这是解决办法之一；二、STC15W201S系列A版本的比较器比较结果标志位(CMPRES)是正确的，因此用户还可用软件查询方式解决该这一问题。

对于上述问题，敬请广大客户留意！对于给各位客户带来的不便，我们深表歉意，请各位客户谅解！我们将在STC15W201S系列的下一版本，即STC15W201S系列B版本中修正这一BUG。

STC MCU Limited

第13章 使用STC15系列单片机的ADC做电容感应触摸按键

按键是电路最常用的零件之一，是人机界面重要的输入方式，我们最熟悉的是机械式按键，但是机械按键有一个缺点（特别是便宜的按键），触点有寿命，很容易出现接触不良而失效。而非接触的按键则没有机械触点，寿命常，使用方便。

非接触的按键有多种方案，而电容感应按键则是低成本的方案，多年前一般是使用专门的IC来实现，随着MCU功能的加强，以及广大用户的实践经验，直接使用MCU来做电容感应按键的技术已经成熟，其中最典型最可靠的是使用ADC做的方案。

本文档详述使用STC带ADC的系列MCU做的方案，可以使用任何带ADC功能的MCU来实现。

下面前3个图是用得最多的方式，原理都一样，本文使用第2个图。

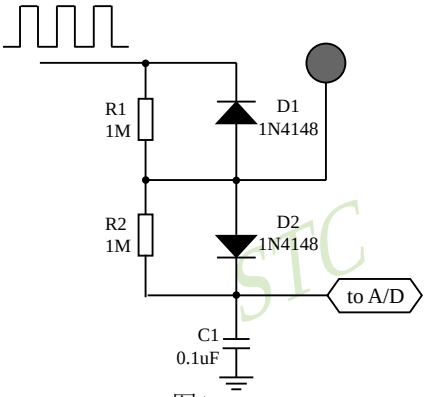


图1

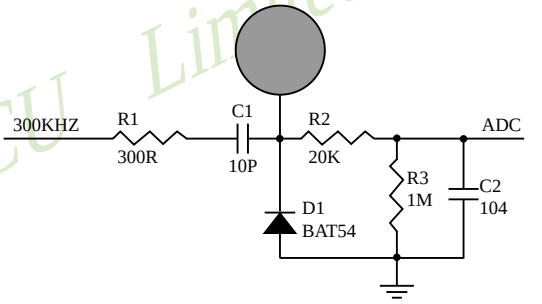


图2

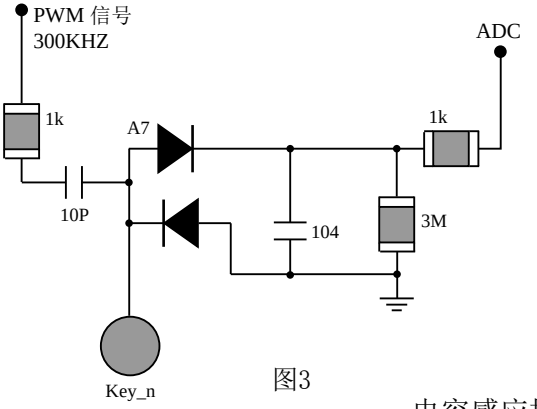


图3

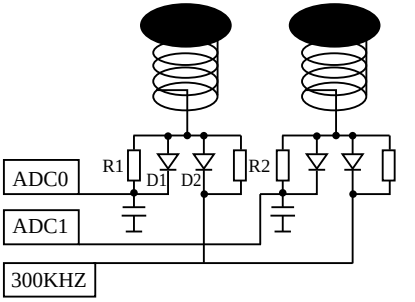
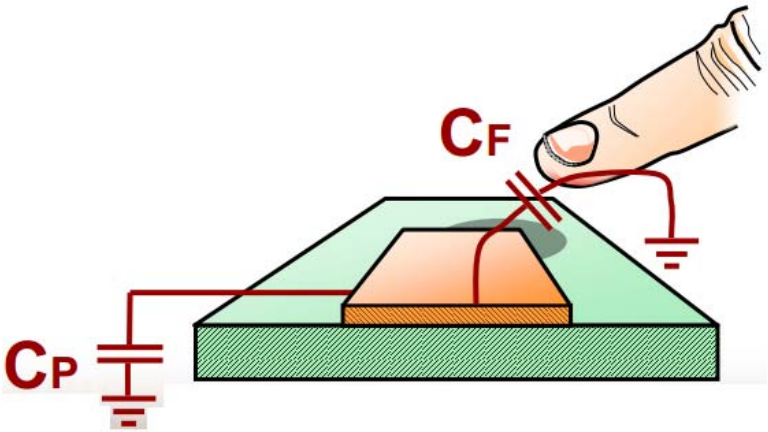


图4 加了感应弹簧

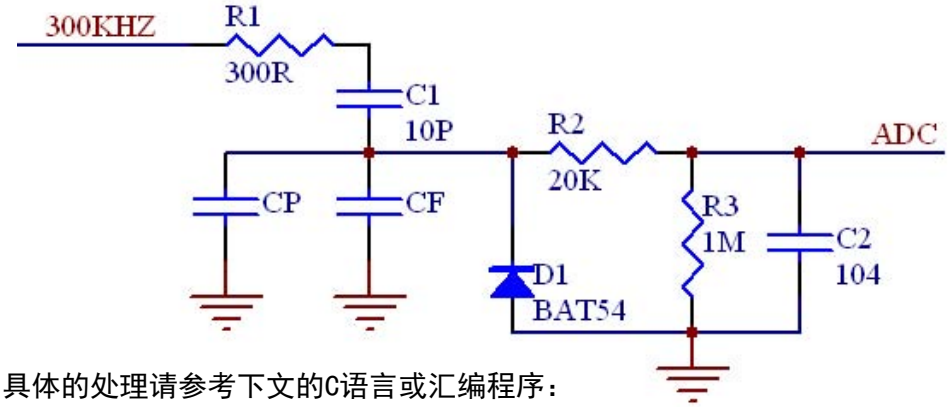
电容感应按键取样电路

STC MCU Limited

一般实际应用时，都使用图4所示的感应弹簧来加大手指按下的面积。感应弹簧等效一块对地的金属板，对地有一个电容CP，而手指按下后，则再并联一个对地的电容CF，如下图所示。



下面为电路图的说明，CP为金属板和分布电容，CF为手指电容，并联在一起与C1对输入的300KHZ方波进行分压，经过D1整流，R2、C2滤波后送ADC，当手指压上去后，送去ADC的电压降低，程序就可以检测出按键动作。



具体的处理请参考下文的C语言或汇编程序：

1、C语言程序

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 ADC做电容触摸按键程序举例----------------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
```

```
/*-----------------------------------------------------------------------------------------------*/

/*************    功能说明          **************

测试使用STC15W408AS的ADC做的电容感应触摸键.

假定测试芯片的工作频率为24MHz

******************************************/

#include  <reg51.h>
#include  <intrins.h>

#define MAIN_Fosc                    24000000UL        //定义主时钟

typedef   unsigned char      u8;
typedef   unsigned int       u16;
typedef   unsigned long      u32;

#define   Timer0_Reload      (65536UL -(MAIN_Fosc / 600000))    //Timer 0 重装值，对应300KHZ

sfr       P1ASF              = 0x9D;          //只写，模拟输入选择
sfr       ADC_CONTR          = 0xBC;          //带AD系列
sfr       ADC_RES            = 0xBD;          //带AD系列
sfr       ADC_RESL           = 0xBE;          //带AD系列
sfr       AUXR               = 0x8E;
sfr       AUXR2              = 0x8F;



/*************    本地常量声明      **************/

#define   TOUCH_CHANNEL          8          //ADC通道数

#define ADC_90T  (3<<5)                     //ADC时间 90T
#define ADC_180T(2<<5)                      //ADC时间 180T
#define ADC_360T(1<<5)                      //ADC时间 360T
#define ADC_540T0                           //ADC时间 540T
#define ADC_FLAG  (1<<4)                    //软件清0
#define ADC_START (1<<3)                    //自动清0

/*************    本地变量声明      **************/
sbit      P_LED7 = P2^7;
sbit      P_LED6 = P2^6;
```

STC MCU Limited

```
sbit    P_LED5 = P2^5;
sbit    P_LED4 = P2^4;
sbit    P_LED3 = P2^3;
sbit    P_LED2 = P2^2;
sbit    P_LED1 = P2^1;
sbit    P_LED0 = P2^0;

u16     idata adc[TOUCH_CHANNEL];                   //当前ADC值
u16     idata adc_prev[TOUCH_CHANNEL];              //上一个ADC值
u16     idata TouchZero[TOUCH_CHANNEL];             //0点ADC值
u8      idata TouchZeroCnt[TOUCH_CHANNEL];          //0点自动跟踪计数

u8      cnt_250ms;

/**************   本地函数声明   **************/
void    delay_ms(u8 ms);
void    ADC_init(void);
u16     Get_ADC10bitResult(u8 channel);
void    AutoZero(void);
u8      check_adc(u8 index);
void    ShowLED(void);

/********************* 主函数 **************************/
void main(void)
{
        u8      i;

        delay_ms(50);

        ET0 = 0;                                    //初始化Timer0输出一个300KHZ时钟
        TR0 = 0;
        AUXR |=  0x80;                              //Timer0 set as 1T mode
        AUXR2 |= 0x01;                              //允许输出时钟
        TMOD = 0;                                   //Timer0 set as Timer, 16 bits Auto Reload.
        TH0 = (u8)(Timer0_Reload >> 8);
        TL0 = (u8)Timer0_Reload;
        TR0 = 1;

        ADC_init();                                 //ADC初始化
        delay_ms(50);                               //延时50ms

        for(i=0; i<TOUCH_CHANNEL; i++)              //初始化0点和上一个值和0点自动跟踪计数
        {
                adc_prev[i]  = 1023;
                TouchZero[i] = 1023;
```

STC MCU Limited

```
                TouchZeroCnt[i] = 0;
        }
        cnt_250ms = 0;

        while (1)
        {
                delay_ms(50);                   //每隔50ms处理一次按键
                ShowLED();
                if(++cnt_250ms >= 5)
                {
                        cnt_250ms = 0;
                        AutoZero();             //每隔250ms处理一次0点自动跟踪
                }

        }
}
/*********************************************/


//=========================================================================
// 函数: void  delay_ms(unsigned char ms)
// 描述: 延时函数。
// 参数: ms,要延时的ms数, 这里只支持1~255ms. 自动适应主时钟.
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//=========================================================================
void  delay_ms(u8 ms)
{
    unsigned int i;
        do {
                i = MAIN_Fosc / 13000;
                while(--i) ;
        }while(--ms);
}

/**************  ADC初始化函数 *****************/
void    ADC_init(void)
{
        P1ASF = 0xff;                           //8路ADC
        ADC_CONTR = 0x80;                       //允许ADC
}

//=========================================================================
```

STC MCU Limited

```
// 函数: u16        Get_ADC10bitResult(u8 channel)
// 描述: 查询法读一次ADC结果.
// 参数: channel: 选择要转换的ADC.
// 返回: 10位ADC结果.
// 版本: V1.0, 2012-10-22
//========================================================================
u16     Get_ADC10bitResult(u8 channel)      //channel = 0~7
{
        ADC_RES  = 0;
        ADC_RESL = 0;
        ADC_CONTR = 0x80 | ADC_90T | ADC_START | channel;          //触发ADC
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        while((ADC_CONTR & ADC_FLAG) == 0)      ;                  //等待ADC转换结束
        ADC_CONTR = 0x80;                                          //清除标志
        return(((u16)ADC_RES << 2) | ((u16)ADC_RESL & 3));         //返回ADC结果
}

/********************** 自动0点跟踪函数 **************************/
void    AutoZero(void)              //250ms调用一次 这是使用相邻2个采样的差的绝对值之和来检测。
{
        u8      i;
        u16     j,k;

        for(i=0; i<TOUCH_CHANNEL; i++)              //处理8个通道
        {
                j = adc[i];
                k = j - adc_prev[i];                //减前一个读数
                F0 = 0;                             //按下
                if(k & 0x8000)      F0 = 1,    k = 0 - k; //释放     求出两次采样的差值
                if(k >= 20)                         //变化比较大
                {
                        TouchZeroCnt[i] = 0;        //如果变化比较大，则清0计数器
                        if(F0)      TouchZero[i] = j;   //如果是释放，并且变化比较大，则直接替代
                }
                else                                //变化比较小，则蠕动，自动0点跟踪
                {
                        if(++TouchZeroCnt[i] >= 20)         //连续检测到小变化20次/4 = 5秒.
                        {
                                TouchZeroCnt[i] = 0;
                                TouchZero[i] = adc_prev[i];  //变化缓慢的值作为0点
```

STC MCU Limited

```
                        }
                }
                adc_prev[i] = j;      //保存这一次的采样值
        }
}


/********************** 获取触摸信息函数 50ms调用1次 ************************/
u8   check_adc(u8 index)                                    //判断键按下或释放,有回差控制
{
        u16     delta;
        adc[index] = 1023 - Get_ADC10bitResult(index);      //获取ADC值, 转成按下键, ADC值增加
        if(adc[index] < TouchZero[index])   return   0;     //比0点还小的值，则认为是键释放
        delta = adc[index] - TouchZero[index];
        if(delta >= 40)     return 1;                       //键按下
        if(delta <= 20)     return 0;                       //键释放
        return   2;                                         //保持原状态
}


/********************** 键处理 50ms调用1次 ************************/
void    ShowLED(void)
{
        u8      i;

        i = check_adc(0);
        if(i == 0)          P_LED0 = 1;     //指示灯灭
        if(i == 1)          P_LED0 = 0;     //指示灯亮

        i = check_adc(1);
        if(i == 0)          P_LED1 = 1;     //指示灯灭
        if(i == 1)          P_LED1 = 0;     //指示灯亮

        i = check_adc(2);
        if(i == 0)          P_LED2 = 1;     //指示灯灭
        if(i == 1)          P_LED2 = 0;     //指示灯亮

        i = check_adc(3);
        if(i == 0)          P_LED3 = 1;     //指示灯灭
        if(i == 1)          P_LED3 = 0;     //指示灯亮

        i = check_adc(4);
        if(i == 0)          P_LED4 = 1;     //指示灯灭
        if(i == 1)          P_LED4 = 0;     //指示灯亮
```

STC MCU Limited

```
        i = check_adc(5);
        if(i == 0)          P_LED5 = 1;     //指示灯灭
        if(i == 1)          P_LED5 = 0;     //指示灯亮

        i = check_adc(6);
        if(i == 0)          P_LED6 = 1;     //指示灯灭
        if(i == 1)          P_LED6 = 0;     //指示灯亮

        i = check_adc(7);
        if(i == 0)          P_LED7 = 1;     //指示灯灭
        if(i == 1)          P_LED7 = 0;     //指示灯亮

}
```

STC MCU Limited

2、汇编程序

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 ADC做电容触摸按键程序举例-------------------------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------*/

;/*************  功能说明            **************

;测试使用STC15W408AS的ADC做的电容感应触摸键.

;假定测试芯片的工作频率为24MHz

;*****************************************/

;/****************************** 用户定义宏 ***********************************/

Fosc_KHZ        EQU     24000                   ;定义主时钟 KHZ

STACK_POIRTER EQU   0D0H                    ;堆栈开始地址

Timer0_Reload   EQU     (65536 - Fosc_KHZ/600)    ;Timer 0 重装值， 对应300KHZ

;*************************************************************************
;*************************************************************************

P1ASF           DATA    0x9D;                   //只写，模拟输入选择
ADC_CONTR       DATA    0xBC;                   //带AD系列
ADC_RES         DATA    0xBD;                   //带AD系列
ADC_RESL        DATA    0xBE;                   //带AD系列
AUXR            DATA    0x8E;
AUXR2           DATA    0x8F;

;*************  本地变量声明      **************/


/*************  本地常量声明      **************/
TOUCH_CHANNEL       EQU     8               ;ADC通道数
```

```
ADC_90T         EQU (3 SHL 5)                   ;ADC时间 90T
ADC_180T        EQU (2 SHL 5)                   ;ADC时间 180T
ADC_360T        EQU (1 SHL 5)                   ;ADC时间 360T
ADC_540T        EQU 0                           ;ADC时间 540T
ADC_FLAG        EQU (1 SHL 4)                   ;软件清0
ADC_START       EQU (1 SHL 3)                   ;自动清0

/*************  本地变量声明    **************/
P_LED7  BIT     P2.7;
P_LED6  BIT     P2.6;
P_LED5  BIT     P2.5;
P_LED4  BIT     P2.4;
P_LED3  BIT     P2.3;
P_LED2  BIT     P2.2;
P_LED1  BIT     P2.1;
P_LED0  BIT     P2.0;

adc             EQU     30H     ;       当前ADC值       30H~3FH, 两字节一个值
adc_prev        EQU     40H     ;       上一个ADC值     40H~4FH, 两字节一个值
TouchZero       EQU     50H     ;       0点ADC值        50H~5FH, 两字节一个值
TouchZeroCnt    EQU     60H     ;       0点自动跟踪计数 60H~67H

cnt_250ms       DATA    68H     ;

;*******************************************************************
;*******************************************************************
                ORG     00H                             ;reset
                LJMP    F_Main

                ORG     03H                             ;0 INT0 interrupt
                RETI
                LJMP    F_INT0_Interrupt

                ORG     0BH                             ;1  Timer0 interrupt
                LJMP    F_Timer0_Interrupt

                ORG     13H                             ;2  INT1 interrupt
                LJMP    F_INT1_Interrupt

                ORG     1BH                             ;3  Timer1 interrupt
                LJMP    F_Timer1_Interrupt

                ORG     23H                             ;4  UART1 interrupt
                LJMP    F_UART1_Interrupt
```

STC MCU Limited

```
                ORG     2BH                     ;5  ADC and SPI interrupt
                LJMP    F_ADC_Interrupt

                ORG     33H                     ;6  Low Voltage Detect interrupt
                LJMP    F_LVD_Interrupt

                ORG     3BH                     ;7  PCA interrupt
                LJMP    F_PCA_Interrupt

                ORG     43H                     ;8  UART2 interrupt
                LJMP    F_UART2_Interrupt

                ORG     4BH                     ;9  SPI interrupt
                LJMP    F_SPI_Interrupt

                ORG     53H                     ;10  INT2 interrupt
                LJMP    F_INT2_Interrupt

                ORG     5BH                     ;11  INT3 interrupt
                LJMP    F_INT3_Interrupt

                ORG     63H                     ;12  Timer2 interrupt
                LJMP    F_Timer2_Interrupt

                ORG     83H                     ;16  INT4 interrupt
                LJMP    F_INT4_Interrupt

;********************* 主程序 ***************************/
F_Main:

        MOV     R0, #1
L_ClearRamLoop:                                 ;清除RAM
        MOV     @R0, #0
        INC     R0
        MOV     A, R0
        CJNE    A, #0FFH, L_ClearRamLoop

        MOV     SP, #STACK_POIRTER
        MOV     PSW, #0
        USING   0                               ;选择第0组R0~R7

;================= 用户初始化程序 ====================================
        MOV     R7, #50
```

STC MCU Limited

```
        LCALL   F_delay_ms

        CLR     ET0                             ; 初始化Timer0输出一个300KHZ时钟
        CLR     TR0                             ;
        ORL     AUXR,  #080H                    ; Timer0 set as 1T mode
        ORL     AUXR2, #01H                     ; 允许输出时钟
        MOV     TMOD,  #0                       ; Timer0 set as Timer, 16 bits Auto Reload.
        MOV     TH0, #HIGH Timer0_Reload
        MOV     TL0, #LOW  Timer0_Reload        ;
        SETB    TR0

        LCALL   F_ADC_init
        MOV     R7, #50
        LCALL   F_delay_ms

        MOV     R0, #adc_prev                   ; 初始化上一个ADC值
L_Init_Loop1:
        MOV     @R0, #03H
        INC     R0
        MOV     @R0, #0FFH
        INC     R0
        MOV     A, R0
        CJNE    A, #(adc_prev + TOUCH_CHANNEL * 2), L_Init_Loop1

        MOV     R0, #TouchZero                  ; 初始化0点ADC值
L_Init_Loop2:
        MOV     @R0, #03H
        INC     R0
        MOV     @R0, #0FFH
        INC     R0
        MOV     A, R0
        CJNE    A, #(TouchZero + TOUCH_CHANNEL * 2), L_Init_Loop2

        MOV     R0, #TouchZeroCnt; 初始化自动跟踪计数值
L_Init_Loop3:
        MOV     @R0, #0
        INC     R0
        MOV     A, R0
        CJNE    A, #(TouchZeroCnt + TOUCH_CHANNEL), L_Init_Loop3

        MOV     cnt_250ms, #5

;================== 主循环 ==================================
L_MainLoop:
```

STC MCU Limited

```
        MOV     R7, #50                 ;延时50ms
        LCALL   F_delay_ms
        LCALL   F_ShowLED               ; 处理一次触摸键值
        DJNZ    cnt_250ms, L_MainLoop

        MOV     cnt_250ms, #5           ;250ms处理一次0点自动跟踪
        LCALL   F_AutoZero              ;自动跟踪零点

        SJMP    L_MainLoop

;==================== 主程序结束 ==================================


; /************* ADC初始化函数 *****************/
F_ADC_init:
        MOV     P1ASF,#0FFH             ;8路ADC
        MOV     ADC_CONTR,#080H         ;允许ADC
        RET
; END OF ADC_init


; //========================================================================
; // 函数: F_Get_ADC10bitResult
; // 描述: 查询法读一次ADC结果.
; // 参数: R7: 选择要转换的ADC.
; // 返回: R6 R7 == 10位ADC结果.
; // 版本: V1.0, 2014-3-25
; //========================================================================

F_Get_ADC10bitResult:
        USING   0                                                       ;选择第0组R0~R7

        MOV     ADC_RES, #0
        MOV     ADC_RESL,#0
        MOV     A, R7
        ORL     A, #0E8H            ;(0x80 OR ADC_90T OR ADC_START)     ;触发ADC
        MOV     ADC_CONTR, A
        NOP
        NOP
        NOP
        NOP

L_10bitADC_Loop1:
        MOV     A, ADC_CONTR
```

STC MCU Limited

```
        JNB     ACC.4, L_10bitADC_Loop1         ;等待ADC转换结束

        MOV     ADC_CONTR,#080H                 //清除标志
        MOV     A,ADC_RES
        MOV     B,#04H
        MUL     AB
        MOV     R7,A
        MOV     R6,B
        MOV     A, ADC_RESL
        ANL     A, #03H
        ORL     A,R7
        MOV     R7,A
        RET
; END OF _Get_ADC10bitResult


; /********************** 自动0点跟踪函数 ************************/
F_AutoZero:                             ;250ms调用一次 这是使用相邻2个采样的差的绝对值之和来检测。
        USING   0                       ;选择第0组R0~R7

        CLR     A
        MOV     R5,A
L_AutoZero_Loop:
                                        ;[R6 R7] = adc[i],  (j = adc[i])
        MOV     A,R5
        ADD     A,ACC
        ADD     A,#LOW (adc)
        MOV     R0,A
        MOV     A,@R0
        MOV     R6,A
        INC     R0
        MOV     A,@R0
        MOV     R7,A

        ; 计算差值 [R2 R3] = adc[i] - adc_prev[i],   (k = j - adc_prev[i];)      //减前一个读数
        MOV     A,R5
        ADD     A,ACC
        ADD     A,#LOW (adc_prev+01H)
        MOV     R0,A
        CLR     C
        MOV     A,R7
        SUBB    A,@R0
        MOV     R3,A
```

STC MCU Limited

```
        MOV     A,R6
        DEC     R0
        SUBB    A,@R0
        MOV     R2,A

        ; 求差值的绝对值 [R2 R3],  if(k & 0x8000)       F0 = 1,    k = 0 - k; //释放     求出两次采样的差值
        CLR     F0                      ;按下
        JNB     ACC.7, L_AutoZero_1
        SETB    F0
        CLR             C
        CLR             A
        SUBB    A, R3
        MOV             R3, A
        MOV     A,R3
        CLR             A
        SUBB    A, R2
        MOV             R2, A

L_AutoZero_1:
        CLR     C                               ;计算 [R2 R3] - #20,  if(k >= 20)        //变化比较大
        MOV     A,R3
        SUBB    A,#20
        MOV     A,R2
        SUBB    A,#00H
        JC      L_AutoZero_2    ;[R2 R3] , 20, 转

        MOV     A,#LOW (TouchZeroCnt)    ;如果变化比较大，则清0计数器       TouchZeroCnt[i] = 0;
        ADD     A,R5
        MOV     R0,A
        MOV     @R0, #0

        ;       if(F0)    TouchZero[i] = j;             //如果是释放，并且变化比较大，则直接替代
        JNB     F0,L_AutoZero_3
        MOV     A,R5
        ADD     A,ACC
        ADD     A,#LOW (TouchZero)
        MOV     R0,A
        MOV     @R0,AR6
        INC     R0
        MOV     @R0,AR7
        SJMP    L_AutoZero_3

L_AutoZero_2:           ;变化比较小，则蠕动，自动0点跟踪
                ;       if(++TouchZeroCnt[i] >= 20)             //连续检测到小变化20次/4 = 5秒.
```

STC MCU Limited

```
        MOV     A,#LOW (TouchZeroCnt)
        ADD     A,R5
        MOV     R0,A
        INC     @R0
        MOV     A,@R0
        CLR     C
        SUBB    A,#20
        JC      L_AutoZero_3        ;if(TouchZeroCnt[i] < 20), 转

        MOV     @R0, #0             ;TouchZeroCnt[i] = 0;

        MOV     A,R5                ;TouchZero[i] = adc_prev[i];  //变化缓慢的值作为0点
        ADD     A,ACC
        ADD     A,#LOW (adc_prev)
        MOV     R0,A
        MOV     A,@R0
        MOV     R2,A
        INC     R0
        MOV     A,@R0
        MOV     R3,A
        MOV     A,R5
        ADD     A,ACC
        ADD     A,#LOW (TouchZero)
        MOV     R0,A
        MOV     @R0,AR2
        INC     R0
        MOV     @R0,AR3

L_AutoZero_3:
                ;           保存采样值          adc_prev[i] = j;
        MOV     A,R5
        ADD     A,ACC
        ADD     A,#LOW (adc_prev)
        MOV     R0,A
        MOV     @R0,AR6
        INC     R0
        MOV     @R0,AR7

        INC     R5
        MOV     A,R5
        XRL     A,#08H
        JZ      $ + 5H
        LJMP    L_AutoZero_Loop
        RET
```

STC MCU Limited

```
; END OF AutoZero

; /********************** 获取触摸信息函数 50ms调用1次 ************************/

F_check_adc:                    ;判断键按下或释放,有回差控制
        USING   0               ;选择第0组R0~R7

        MOV     R4, AR7
;       adc[index] = 1023 - Get_ADC10bitResult(index);          //获取ADC值, 转成按下键, ADC值增加
        LCALL   F_Get_ADC10bitResult    ;返回的ADC值在 [R6 R7]
        CLR     C
        MOV     A,#0FFH         ;1023 - [R6 R7]
        SUBB    A,R7
        MOV     R7,A
        MOV     A,#03H
        SUBB    A,R6
        MOV     R6,A

        MOV     A,R4            ;保存 adc[index]
        ADD     A,ACC
        ADD     A,#LOW (adc)
        MOV     R0,A
        MOV     @R0,AR6
        INC     R0
        MOV     @R0,AR7

;       if(adc[index] < TouchZero[index])       return      0;          //比0点还小的值，则认为是键释放
        MOV     A,R4
        ADD     A,ACC
        ADD     A,#LOW (TouchZero+01H)
        MOV     R1,A
        MOV     A,R4
        ADD     A,ACC
        ADD     A,#LOW (adc)
        MOV     R0,A
        MOV     A,@R0
        MOV     R6,A
        INC     R0
        MOV     A,@R0
        CLR     C
        SUBB    A,@R1           ;计算 adc[index] - TouchZero[index]
        MOV     A,R6
        DEC     R1
        SUBB    A,@R1
```

STC MCU Limited

```
        JNC     L_check_adc_1   ;if(adc[index] >= TouchZero[index]), 转
        MOV     R7,#00H         ;if(adc[index] < TouchZero[index]), 比0点还小的值,
                                ;则认为是键释放, 返回0
        RET

L_check_adc_1:
                                ; 计算差值          [R6 R7] = delta = adc[index] - TouchZero[index];
        MOV     A,R4
        ADD     A,ACC
        ADD     A,#LOW (TouchZero+01H)
        MOV     R1,A
        MOV     A,R4
        ADD     A,ACC
        ADD     A,#LOW (adc+01H)
        MOV     R0,A
        CLR     C
        MOV     A,@R0
        SUBB    A,@R1
        MOV     R7,A
        DEC     R0
        MOV     A,@R0
        DEC     R1
        SUBB    A,@R1
        MOV     R6,A

                                ;---- Variable 'delta' assigned to Register 'R6/R7' ----
        CLR     C
        MOV     A,R7
        SUBB    A,#40
        MOV     A,R6
        SUBB    A,#00H
        JC      L_check_adc_2   ;if(delta < 40), 转
        MOV     R7,#1           ;if(delta >= 40)      return 1;  //键按下  返回1
        RET

L_check_adc_2:
        SETB    C
        MOV     A,R7
        SUBB    A,#20
        MOV     A,R6
        SUBB    A,#00H
        JNC     L_check_adc_3
        MOV     R7,#0           ;if(delta <= 20)      return 0;  //键释放  返回0
        RET
```

STC MCU Limited

```
L_check_adc_3:
        MOV     R7,#2           ;if((delta > 20) && (delta < 40))       保持原状态  返回2
        RET
; END OF _check_adc

/********************** 键处理 50ms调用1次 ************************/

F_ShowLED:
        USING   0               ;选择第0组R0~R7

        MOV     R7, #0
        LCALL   F_check_adc
        MOV     A,R7
        ANL     A, #0FEH
        JNZ     L_QuitCheck0
        MOV     A, R7
        MOV     C, ACC.0
        CPL     C
        MOV     P_LED0, C       ;if(i == 0) 指示灯灭, if(i == 1) 指示灯亮
L_QuitCheck0:

        MOV     R7, #1
        LCALL   F_check_adc
        MOV     A,R7
        ANL     A, #0FEH
        JNZ     L_QuitCheck1
        MOV     A, R7
        MOV     C, ACC.0
        CPL     C
        MOV     P_LED1, C       ;if(i == 0) 指示灯灭, if(i == 1) 指示灯亮
L_QuitCheck1:

        MOV     R7, #2
        LCALL   F_check_adc
        MOV     A,R7
        ANL     A, #0FEH
        JNZ     L_QuitCheck2
        MOV     A, R7
        MOV     C, ACC.0
        CPL     C
        MOV     P_LED2, C       ;if(i == 0) 指示灯灭, if(i == 1) 指示灯亮
L_QuitCheck2:

        MOV     R7, #3
        LCALL   F_check_adc
```

STC MCU Limited

```
        MOV     A,R7
        ANL     A, #0FEH
        JNZ     L_QuitCheck3
        MOV     A, R7
        MOV     C, ACC.0
        CPL     C
        MOV     P_LED3, C       ;if(i == 0) 指示灯灭, if(i == 1) 指示灯亮
L_QuitCheck3:

        MOV     R7, #4
        LCALL   F_check_adc
        MOV     A,R7
        ANL     A, #0FEH
        JNZ     L_QuitCheck4
        MOV     A, R7
        MOV     C, ACC.0
        CPL     C
        MOV     P_LED4, C       ;if(i == 0) 指示灯灭, if(i == 1) 指示灯亮
L_QuitCheck4:

        MOV     R7, #5
        LCALL   F_check_adc
        MOV     A,R7
        ANL     A, #0FEH
        JNZ     L_QuitCheck5
        MOV     A, R7
        MOV     C, ACC.0
        CPL     C
        MOV     P_LED5, C       ;if(i == 0) 指示灯灭, if(i == 1) 指示灯亮
L_QuitCheck5:

        MOV     R7, #6
        LCALL   F_check_adc
        MOV     A,R7
        ANL     A, #0FEH
        JNZ     L_QuitCheck6
        MOV     A, R7
        MOV     C, ACC.0
        CPL     C
        MOV     P_LED6, C       ;if(i == 0) 指示灯灭, if(i == 1) 指示灯亮
L_QuitCheck6:

        MOV     R7, #7
        LCALL   F_check_adc
```

STC MCU Limited

```
        MOV     A,R7
        ANL     A, #0FEH
        JNZ     L_QuitCheck7
        MOV     A, R7
        MOV     C, ACC.0
        CPL     C
        MOV     P_LED7, C           ;if(i == 0) 指示灯灭, if(i == 1) 指示灯亮
L_QuitCheck7:

        RET
; END OF ShowLED

;//==========================================================================
;// 函数: F_delay_ms
;// 描述: 延时子程序。
;// 参数: R7: 延时ms数.
;// 返回: none.
;// 版本: VER1.0
;// 日期: 2013-4-1
;// 备注: 除了ACCC和PSW外, 所用到的通用寄存器都入栈
;//==========================================================================
F_delay_ms:
        PUSH    AR3                 ;入栈R3
        PUSH    AR4                 ;入栈R4

L_delay_ms_1:
        MOV     R3, #HIGH (Fosc_KHZ / 13)
        MOV     R4, #LOW (Fosc_KHZ / 13)

L_delay_ms_2:
        MOV     A, R4               ;1T             Total 13T/loop
        DEC     R4                  ;2T
        JNZ     L_delay_ms_3        ;4T
        DEC     R3
L_delay_ms_3:
        DEC     A                   ;1T
        ORL     A, R3               ;1T
        JNZ     L_delay_ms_2        ;4T

        DJNZ    R7, L_delay_ms_1

        POP     AR4             ;出栈R2
        POP     AR3             ;出栈R3
        RET
```

STC MCU Limited

```
;*******************************************************************
;***************** 中断函数 ****************************************************
F_Timer0_Interrupt:
        RETI

F_Timer1_Interrupt:
        RETI

F_Timer2_Interrupt:
        RETI

F_INT0_Interrupt:
        RETI

F_INT1_Interrupt:
        RETI

F_INT2_Interrupt:
        RETI

F_INT3_Interrupt:
        RETI

F_INT4_Interrupt:
        RETI

F_UART1_Interrupt:
        RETI

F_UART2_Interrupt:
        RETI

F_ADC_Interrupt:
        RETI

F_LVD_Interrupt:
        RETI

F_PCA_Interrupt:
        RETI

F_SPI_Interrupt:
        RETI

        END
```

STC MCU Limited

第14章 同步串行外围接口(SPI接口)

STC15系列单片机还提供另一种高速串行通信接口 —— SPI接口。SPI是一种全双工、高速、同步的通信总线，有两种操作模式：主模式和从模式。在主模式中支持高达3 Mbps的速率(工作频率为12MHz时,如果CPU主频采用20MHz到36MHz,则可更高，从模式时速度无法太快，SYSclk/4以内较好),还具有传输完成标志和写冲突标志保护。

下表总结了STC15系列单片机内部集成了SPI功能的单片机型号：

特殊外围设备 / 单片机型号	8路10位高速A/D转换器	CCP/PCA/PWM功能	1组高速同步串行口SPI
STC15W4K60S4系列	√	√	√
STC15F2K60S2系列	√	√	√
STC15W1K16S系列			√
STC15W408S系列			√
STC15W408AS系列	√	√	√
STC15W201S系列			
STC15F408AD系列	√	√	√
STC15F101W系列			

上表中 √ 表示对应的系列有相应的功能。

STC MCU Limited

STC15W4K60S4系列、STC15F2K60S2系列、STC15W1K16S系列和STC15W408S系列单片机的SPI可以在3组不同管脚之间进行切换：

[SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5];
[SS_2/P2.4, MOSI_2/P2.3, MISO_2/P2.2, SCLK_2/P2.1];
[SS_3/P5.4, MOSI_3/P4.0, MISO_3/P4.1, SCLK_3/P4.3]。

注意：现STC15F2K60S2及STC15L2K60S2系列C版本的SPI工作于主模式时正常，但工作于从模式时有问题，建议不要使用该版本的SPI从模式。

STC15F408AD系列和STC15W408AS系列单片机的SPI可以在2组不同管脚之间进行切换：

[SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5];
[SS_2/P2.4, MOSI_2/P2.3, MISO_2/P2.2, SCLK_2/P2.1]。

注意：现STC15F408AD及STC15L408AD系列C版本的SPI工作于主模式时正常，但工作于从模式时有问题，建议不要使用该版本的SPI从模式。

STC15W201S系列和STC15F101W系列单片机没有SPI功能。

特别声明：以15F和15L开头且有SPI功能的芯片,只支持"SPI主机模式",不支持"SPI从机模式"；
以15W开头且有SPI功能的芯片，SPI主/从机模式均支持

14.1 与SPI功能模块相关的特殊功能寄存器

STC15系列 1T 8051单片机SPI功能模块特殊功能寄存器 SPI Management SFRs

符号	描述	地址	位地址及其符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SPCTL	SPI Control Register	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	0000,0100
SPSTAT	SPI Status Register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPDAT	SPI Data Register	CFH									0000,0000
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	Interrupt Enable 2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000
IP2	Interrupt Priority	B5H	-	-	-	-	-	-	PSPI	PS2	xxxx,xx00
AUXR1 P_SW1	Auxiliary Register 1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	-	DPS	0000,0000

1. SPI控制寄存器SPCTL

SPI控制寄存器的格式如下：

SPCTL：SPI控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	name	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

SSIG：$\overline{SS}$引脚忽略控制位。

SSIG=1，MSTR（位4）确定器件为主机还是从机；

SSIG=0，$\overline{SS}$脚用于确定器件为主机还是从机. $\overline{SS}$脚可作为I/O口使用(见SPI主从选择表)

SPEN：SPI使能位。

SPEN=1，SPI使能；

SPEN=0，SPI被禁止，所有SPI引脚都作为I/O口使用。

DORD：设定SPI数据发送和接收的位顺序。

DORD=1，数据字的LSB(最低位)最先发送；

DORD=0，数据字的MSB(最高位)最先发送。

MSTR：主/从模式选择位（见SPI主从选择表）。

CPOL：SPI时钟极性。

CPOL=1，SCLK空闲时为高电平。SCLK的前时钟沿为下降沿而后沿为上升沿。

CPOL=0，SCLK空闲时为低电平。SCLK的前时钟沿为上升沿而后沿为下降沿。

CPHA：SPI时钟相位选择。

CPHA=1，数据在SCLK的前时钟沿驱动，并在后时钟沿采样。

CPHA=0，数据在$\overline{SS}$为低（SSIG＝0）时被驱动，在SCLK的后时钟沿被改变，并在前时钟沿被采样。（注：SSIG = 1时的操作未定义）

STC MCU Limited

SPR1、SPR0：SPI时钟速率选择控制位。SPI时钟选择如下表所列。

SPI时钟频率的选择

SPR1	SPR0	时钟(SCLK)
0	0	CPU_CLK/4
0	1	CPU_CLK/16
1	0	CPU_CLK/64
1	1	CPU_CLK/128

其中，CPU_CLK是CPU时钟。

STC MCU Limited

2. SPI状态寄存器SPSTAT

SPI状态寄存器的格式如下：

SPSTAT: SPI状态寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	name	SPIF	WCOL	-	-	-	-	-	-

SPIF：SPI传输完成标志。

当一次串行传输完成时，SPIF置位。此时，如果SPI中断被打开(即ESPI (IE2.1) 和EA(IE.7) 都置位)，则产生中断。当SPI处于主模式且SSIG=0时，如果$\overline{SS}$为输入并被驱动为低电平，SPIF也将置位, 表示“模式改变”。SPIF标志通过软件向其写入″1″清零。

WCOL：SPI写冲突标志。

在数据传输的过程中如果对SPI 数据寄存器SPDAT执行写操作，WCOL将置位。WCOL标志通过软件向其写入″1″清零。

3. SPI数据寄存器SPDAT

SPI数据寄存器的格式如下：

SPDAT: SPI数据寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH	name								

SPDAT.7 - SPDAT.0：传输的数据位Bit7～Bit0

4. 中断允许寄存器IE及IE2

IE：中断允许寄存器 (可位寻址)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	name	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA：CPU的中断开放标志
　　EA=1，CPU开放中断，
　　EA=0，CPU屏蔽所有的中断申请。

EA的作用是使中断允许形成多级控制。即各中断源首先受EA控制;其次还受各中断源自己的中断允许控制位控制。

IE2：中断允许寄存器2

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ESPI：SPI中断允许位
　　ESPI=1，允许SPI中断，
　　ESPI=0，禁止SPI中断。

5. 中断优先级控制寄存器IP2

IP2：中断优先级控制寄存器2

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP2	B5H	name	-	-	-	-	-	-	PSPI	PS2

PSPI: SPI中断优先级控制位。
　　当PSPI=0时，SPI中断为最低优先级中断(优先级0)
　　当PSPI=1时，SPI中断为最高优先级中断(优先级1)

STC MCU Limited

6. 控制SPI功能切换的寄存器AUXR1(P_SW1)

外围设备切换控制寄存器1的格式如下：

AUXR1/P_SW1：外围设备切换控制寄存器1（不可位寻址）

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000,0000

SPI可在3个地方切换，由 SPI_S1 / SPI_S0 两个控制位来选择		
SPI_S1	SPI_S0	SPI可在P1/P2/P4之间来回切换
0	0	SPI在[P1.2/SS,P1.3/MOSI,P1.4/MISO,P1.5/SCLK]
0	1	SPI在[P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI在[P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	无效

CCP可在3个地方切换，由 CCP_S1 / CCP_S0 两个控制位来选择		
CCP_S1	CCP_S0	CCP可在P1/P2/P3之间来回切换
0	0	CCP在[P1.2/ECI,P1.1/CCP0,P1.0/CCP1,P3.7/CCP2]
0	1	CCP在[P3.4/ECI_2,P3.5/CCP0_2,P3.6/CCP1_2,P3.7/CCP2_2]
1	0	CCP在[P2.4/ECI_3,P2.5/CCP0_3,P2.6/CCP1_3,P2.7/CCP2_3]
1	1	无效

串口1/S1可在3个地方切换，由 S1_S0 及 S1_S1 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD,P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2,P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1] 串口1在P1口时要使用内部时钟
1	1	无效

DPS：DPTR registers select bit. DPTR 寄存器选择位

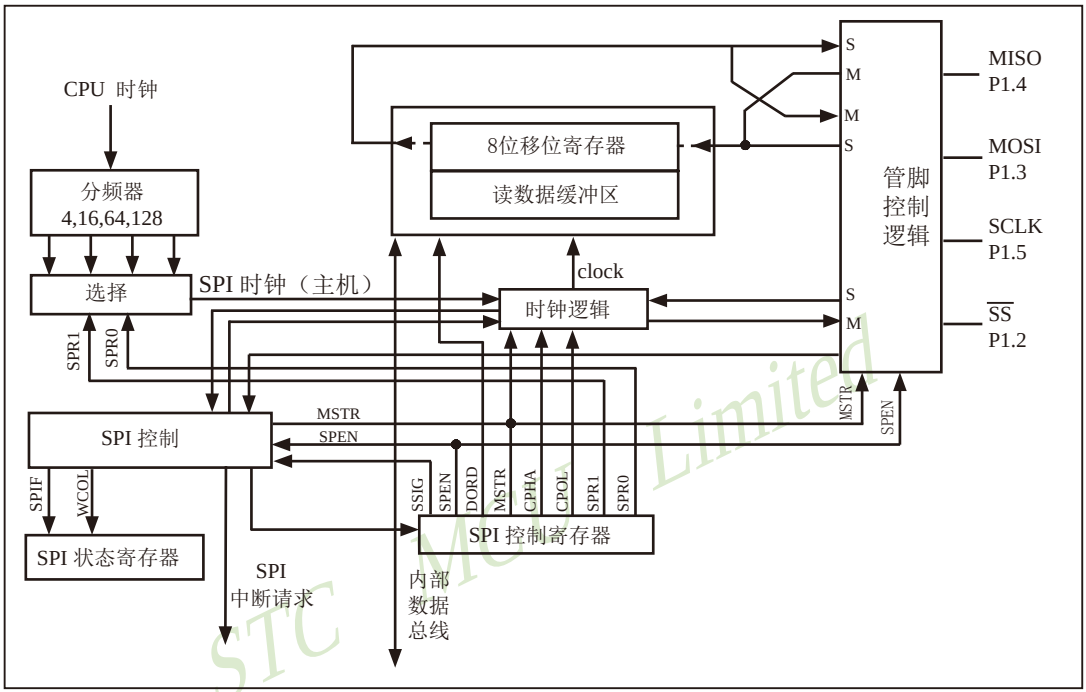
0，使用缺省数据指针DPTR0

1，使用另一个数据指针DPTR1

STC MCU Limited

14.2 SPI接口的结构

STC15系列单片机的SPI功能方框图如下图所示。



SPI 功能方框图

SPI的核心是一个8位移位寄存器和数据缓冲器，数据可以同时发送和接收。在SPI数据的传输过程中，发送和接收的数据都存储在数据缓冲器中。

对于主模式，若要发送一字节数据，只需将这个数据写到SPDAT寄存器中。主模式下$\overline{SS}$信号不是必需的；但是在从模式下，必须在$\overline{SS}$信号变为有效并接收到合适的时钟信号后，方可进行数据传输。在从模式下，如果一个字节传输完成后，$\overline{SS}$信号变为高电平，这个字节立即被硬件逻辑标志为接收完成，SPI接口准备接收下一个数据。

14.3 SPI接口的数据通信

SPI接口有4个管脚：SCLK, MISO, MOSI和$\overline{SS}$，可在3组管脚之间进行切换：[SCLK/P1.5, MISO/P1.4, MOSI/P1.3和$\overline{SS}$/P1.2]；[SCLK_2/P2.1, MISO_2/P2.2, MOSI_2/P2.3和$\overline{SS}$_2/P2.4]；[SCLK_3/P4.3, MISO_3/P4.1, MOSI_3/P4.0和$\overline{SS}$_3/P5.4]。

MOSI (Master Out Slave In，主出从入)：主器件的输出和从器件的输入，用于主器件到从器件的串行数据传输。当SPI作为主器件时，该信号是输出；当SPI作为从器件时，该信号是输入。数据传输时最高位在先，低位在后。根据SPI规范，多个从机可以共享一根MOSI信号线。在时钟边界的前半周期，主机将数据放在MOSI信号线上，从机在该边界处获取该数据。

MISO (Master In Slave Out，主入从出)：从器件的输出和主器件的输入，用于实现从器件到主器件的数据传输。当SPI作为主器件时，该信号是输入；当SPI作为从器件时，该信号是输出。数据传输时最高位在先，低位在后。SPI规范中，一个主机可连接多个从机，因此，主机的MISO信号线会连接到多个从机上，或者说，多个从机共享一根MISO信号线。当主机与一个从机通信时，其他从机应将其MISO引脚驱动置为高阻状态。

SCLK (SPI Clock，串行时钟信号)：串行时钟信号是主器件的输出和从器件的输入，用于同步主器件和从器件之间在MOSI和MISO线上的串行数据传输。当主器件启动一次数据传输时，自动产生8个SCLK时钟周期信号给从机。在SCLK的每个跳变处(上升沿或下降沿)移出一位数据。所以，一次数据传输可以传输一个字节的数据。

SCLK、MOSI和MISO通常和两个或更多SPI器件连接在一起。数据通过MOSI由主机传送到从机，通过MISO由从机传送到主机。SCLK信号在主模式时为输出，在从模式时为输入。如果SPI系统被禁止，即SPEN(SPCTL.6)=0(复位值)，这些管脚都可作为I/O口使用。

$\overline{SS}$(Slave Select，从机选择信号)：这是一个输入信号，主器件用它来选择处于从模式的SPI模块。主模式和从模式下，$\overline{SS}$的使用方法不同。在主模式下，SPI接口只能有一个主机，不存在主机选择问题，该模式下$\overline{SS}$不是必需的。主模式下通常将主机的$\overline{SS}$管脚通过10KΩ的电阻上拉高电平。每一个从机的$\overline{SS}$接主机的I/O口，由主机控制电平高低，以便主机选择从机。在从模式下，不管发送还是接收，$\overline{SS}$信号必须有效。因此在一次数据传输开始之前必须将$\overline{SS}$为低电平。SPI主机可以使用I/O口选择一个SPI器件作为当前的从机。

SPI从器件通过其$\overline{SS}$脚确定是否被选择。如果满足下面的条件之一，$\overline{SS}$就被忽略：

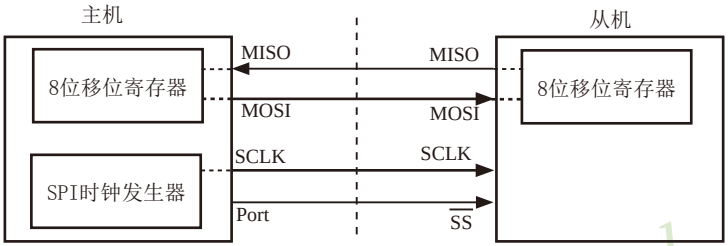
- 如果SPI系统被禁止，即SPEN(SPCTL.6)= 0(复位值)
- 如果SPI配置为主机，即MSTR(SPCTL.4)=1, 并且P1.2/$\overline{SS}$配置为输出（通过P1M0.2和P1M1.2）
- 如果$\overline{SS}$脚被忽略，即SSIG(SPCTL.7)＝ 1，该脚配置用于I/O口功能。

注：即使SPI被配置为主机（MSTR ＝ 1），它仍然可以通过拉低$\overline{SS}$脚配置为从机（如果P1.2/$\overline{SS}$配置为输入且SSIG=0）。要使能该特性，应当置位SPIF(SPSTAT.7)。

14.3.1 SPI接口的数据通信方式

STC15系列单片机的SPI接口的数据通信方式有3种：单主机—从机方式、双器件方式(器件可互为主机和从机)和单主机—多从机方式。

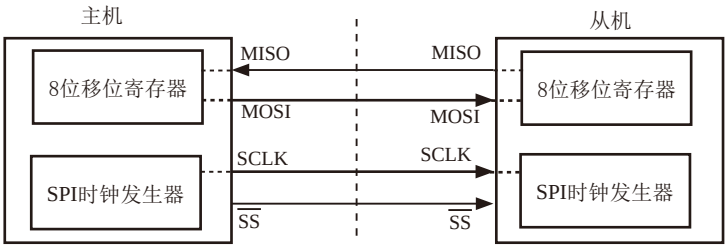
单主机—单从机方式的连接图如下SPI图1所示。



SPI图1　SPI单主机—单从机 配置

在上图SPI图1中，从机的SSIG(SPCTL.7)为0，$\overline{SS}$用于选择从机。SPI主机可使用任何端口（包括P1.2/$\overline{SS}$）来驱动$\overline{SS}$脚。主机SPI与从机SPI的8位移位寄存器连接成一个循环的16位移位寄存器。当主机程序向SPDAT寄存器写入一个字节时，立即启动一个连续的8位移位通信过程：主机的SCLK引脚向从机的SCLK引脚发出一串脉冲，在这串脉冲的驱动下，主机SPI的8位移位寄存器中的数据移动到了从机SPI的8移位寄存器中。与此同时，从机SPI的8位移位寄存器中的数据移动到了主机SPI的8位移位寄存器中。由此，主机既可向从机发送数据，又可读从机中的数据。

双器件方式(器件可互为主机和从机)的连接图如下SPI图2所示。



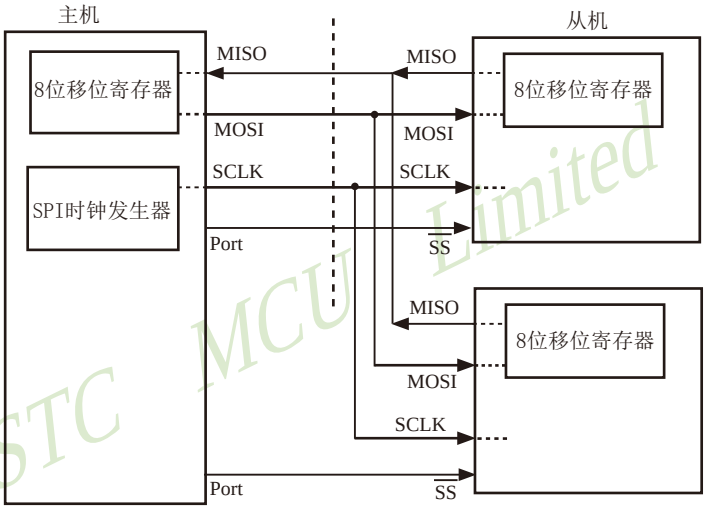
SPI图2　SPI双器件配置(器件可互为主从)

上图SPI图2所示为两个器件互为主从的情况。当没有发生SPI操作时，两个器件都可配置为主机(MSTR=1)，将SSIG清零并将P1.2($\overline{SS}$)配置为准双向模式。当其中一个器件启动传输时，它可将P1.2/$\overline{SS}$配置为输出并驱动为低电平，这样就强制另一个器件变为从机。

双方初始化时将自己设置成忽略$\overline{SS}$脚的SPI从模式。当一方要主动发送数据时，先检测$\overline{SS}$脚的电平，如果$\overline{SS}$脚是高电平，就将自己设置成忽略$\overline{SS}$脚的主模式。通信双方平时将SPI设置成没有被选中的从模式。在该模式下，MISO、MOSI、SCLK均为输入，当多个MCU的SPI接口以此模式并联时不会发生总线冲突。这种特性在互为主/从、一主多从等应用中很有用。

注意：互为主/从模式时，双方的SPI速率必须相同。如果使用外部晶体振荡器，双方的晶体频率也要相同。

双器件方式(器件可互为主机和从机)的连接图如下SPI图3所示。



SPI图3　SPI 单主机-多从机 配置

在上图SPI图3 中，从机的SSIG(SPCTL.7)为0，从机通过对应的$\overline{SS}$信号被选中。SPI主机可使用任何端口(包括P1.2/$\overline{SS}$)来驱动$\overline{SS}$脚。

14.3.2 对SPI进行配置

STC15系列单片机进行SPI通信时，主机和从机的选择由SPEN、SSIG、$\overline{SS}$引脚(P1.2)和MSTR联合控制。下表所示为主/从模式的配置以及模式的使用和传输方向。

SPI 主从模式选择

SPEN	SSIG	$\overline{SS}$脚 P1.2	MSTR	主或从模式	MISO P1.4	MOSI P1.3	SCLK P1.5	备注
0	X	P1.2/$\overline{SS}$	X	SPI功能禁止	P1.4/MISO	P1.3/MOSI	P1.5/SCLK	SPI禁止。P1.2/$\overline{SS}$, P1.3/MOSI, P1.4/MISO和P1.5/SCLK作为普通I/O口使用
1	0	0	0	从机模式	输出	输入	输入	选择作为从机
1	0	1	0	从机模式未被选中	高阻	输入	输入	未被选中。MISO为高阻状态，以避免总线冲突
1	0	0	1—>0	从机模式	输出	输入	输入	P1.2/$\overline{SS}$配置为输入或准双向口。SSIG为0。如果选择$\overline{SS}$被驱动为低电平，则被选择作为从机。当$\overline{SS}$变为低电平时，MSTR将清零。 注：当$\overline{SS}$处于输入模式时，如被驱动为低电平且SSIG=0时，MSTR位自动清零。
1	0	1	1	主(空闲)	输入	高阻	高阻	当主机空闲时MOSI和SCLK为高阻态以避免总线冲突。用户必须将SCLK上拉或下拉（根据CPOL/SPCTL.3 的取值）以避免SCLK出现悬浮状态。
				主(激活)		输出	输出	作为主机激活时，MOSI和SCLK为推挽输出
1	1	P1.2/$\overline{SS}$	0	从	输出	输入	输入	
1	1	P1.2/$\overline{SS}$	1	主	输入	输出	输出	

STC MCU Limited

14.3.3 作为主机/从机时的额外注意事项

作为从机时的额外注意事项

当CPHA＝0时，SSIG必须为0（也就是不能忽略$\overline{SS}$脚），$\overline{SS}$脚必须置低并且在每个连续的串行字节发送完后须重新设置为高电平。如果SPDAT寄存器在$\overline{SS}$有效（低电平）时执行写操作，那么将导致一个写冲突错误。CPHA=0且SSIG=0时的操作未定义。

当CPHA＝1时，SSIG可以置1（即可以忽略$\overline{SS}$脚）。如果SSIG＝0，$\overline{SS}$脚可在连续传输之间保持低有效（即一直固定为低电平）。这种方式有时适用于具有单固定主机和单从机驱动MISO数据线的系统。

作为主机时的额外注意事项

在SPI中，传输总是由主机启动的。如果SPI使能（SPEN=1）并选择作为主机，主机对SPI数据寄存器的写操作将启动SPI时钟发生器和数据的传输。在数据写入SPDAT之后的半个到一个SPI位时间后，数据将出现在MOSI脚。

需要注意的是，主机可以通过将对应器件的$\overline{SS}$脚驱动为低电平实现与之通信。写入主机SPDAT寄存器的数据从MOSI脚移出发送到从机的MOSI脚。同时从机SPDAT寄存器的数据从MISO脚移出发送到主机的MISO脚。

传输完一个字节后，SPI时钟发生器停止，传输完成标志（SPIF）置位并产生一个中断（如果SPI中断使能）。主机和从机CPU的两个移位寄存器可以看作是一个16位循环移位寄存器。当数据从主机移位传送到从机的同时，数据也以相反的方向移入。这意味着在一个移位周期中，主机和从机的数据相互交换。

STC MCU Limited

14.3.4 通过$\overline{SS}$改变模式

如果SPEN=1, SSIG=0且MSTR=1，SPI使能为主机模式。$\overline{SS}$脚可配置为输入([P2M1.2, P2M0.2]=[1,0])或准双向模式([P2M1.2, P2M0.2]=[0,0])。这种情况下，另外一个主机可将该$\overline{SS}$脚驱动为低电平，从而将该器件选择为SPI从机并向其发送数据。

为了避免争夺总线，SPI系统执行以下动作:

1) MSTR清零并且CPU变成从机。这样SPI就变成从机。MOSI和SCLK强制变为输入模式，而MISO则变为输出模式。

2) SPSTAT的SPIF标志位置位。如果SPI中断已被使能，则产生SPI中断。

用户软件必须一直对MSTR位进行检测，如果该位被一个从机选择所清零而用户想继续将SPI作为主机，这时就必须重新置位MSTR，否则就进入从机模式。

STC MCU Limited

14.3.5 写冲突

SPI在发送时为单缓冲，在接收时为双缓冲。这样在前一次发送尚未完成之前，不能将新的数据写入移位寄存器。当发送过程中对数据寄存器进行写操作时，WCOL位(SPSTAT.6)将置位以指示数据冲突。在这种情况下，当前发送的数据继续发送，而新写入的数据将丢失。

当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

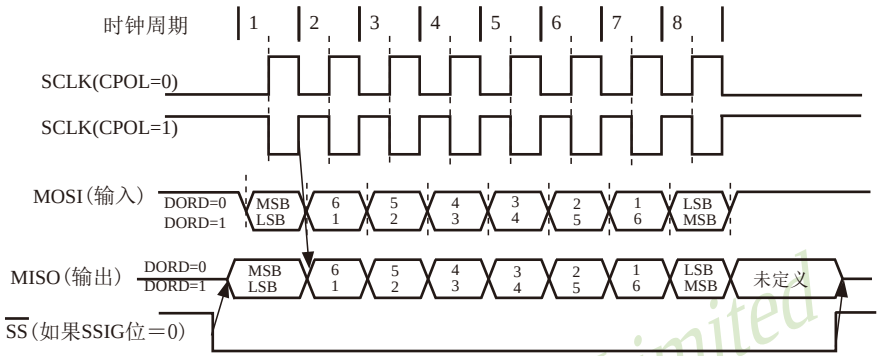
接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收数据将丢失。

WCOL可通过软件向其写入“1”清零。

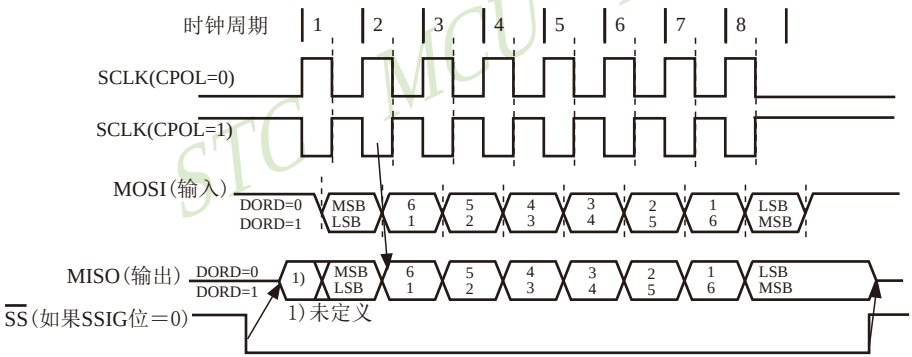
14.3.6 数据模式

时钟相位位(CPHA)允许用户设置采样和改变数据的时钟边沿。时钟极性位CPOL允许用户设置时钟极性。

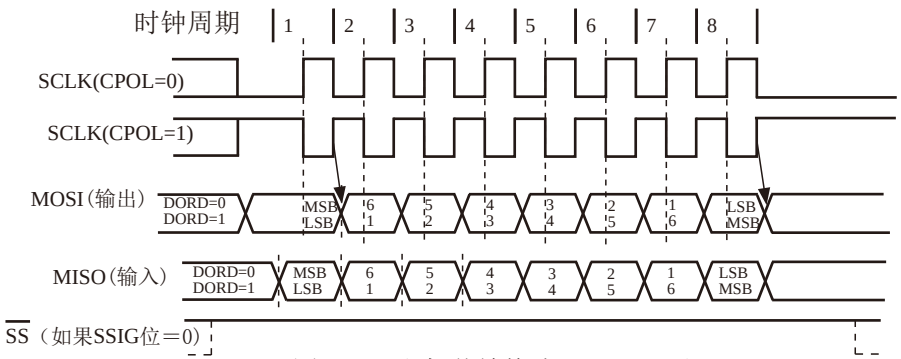
SPI图4～图7 所示为时钟相位位CPHA的不同设定。



SPI图4　SPI 从机传输格式（CPHA=0）

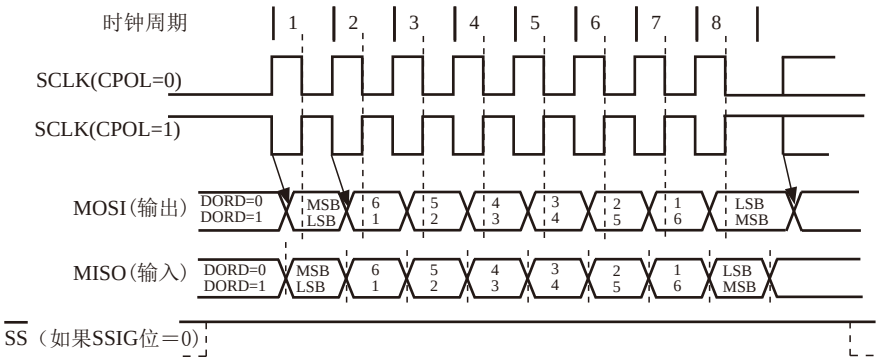


SPI图5　SPI从机传输格式（CPHA=1）



SPI图6　SPI主机传输格式（CPHA=0）

STC MCU Limited



SPI图7　SPI主机传输格式（CPHA=1）

SPI接口的时钟信号线SCLK有Idle和Active两种状态：Idle状态时指在不进行数据传输的时候(或数据传输完成后)SCLK所处的状态；Active是与Idle相对的一种状态。

时钟相位位(CPHA)允许用户设置采样和改变数据的时钟边沿。时钟极性CPOL允许用户设置时钟极性。

如果CPOL=0,Idle状态=低电平，Active状态=高电平；

如果CPOL=1,Idle状态=高电平，Active状态=低电平。

主机总是在SCLK=Idle状态时，将下一位要发送的数据置于数据线MOSI上。

从Idle状态到Active状态的转变，称为SCLK前沿；从Active状态到Idle状态的转变，称为SCLK后沿。一个SCLK前沿和后沿构成一个SCLK时钟周期，一个SCLK时钟周期传输一位数据。

STC-MCU Limited

SPI时钟预分频器选择

SPI时钟预分频器选择是通过SPCTL寄存器中的SPR1-SPR0位实现的

SPI时钟频率的选择

SPR1	SPR0	时钟(SCLK)
0	0	CPU_CLK/4
0	1	CPU_CLK/16
1	0	CPU_CLK/64
1	1	CPU_CLK/128

其中，CPU_CLK是CPU时钟。

14.4 适用单主单从系统的SPI功能测试程序(C和汇编)

14.4.1 中断方式

1. C程序

```
/*---------------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ------------------------------------------------------------*/
/* --- 演示STC 1T 系列单片机  SPI功能(适用单主单从，中断方式)----------------------*/
/* --- 研发顾问QQ：800003751-----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------------ */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------------ */
/* --- Web: www.STCMCU.com ------------------------------------------------------------------ */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序---------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序---------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*---------------------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

//#define  MASTER                  //define:master undefine:slave
#define   FOSC    18432000L
#define   BAUD    (256 - FOSC / 32 / 115200)

typedef unsigned char   BYTE;
typedef unsigned int    WORD;
typedef unsigned long   DWORD;

#define   URMD   0                  //0:使用定时器2作为波特率发生器
                                    //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                    //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器

sfr       T2H     =     0xd6;       //定时器2高8位
sfr       T2L     =     0xd7;       //定时器2低8位

sfr       AUXR    =     0x8e;       //辅助寄存器
sfr       SPSTAT  =     0xcd;       //SPI状态寄存器
#define   SPIF          0x80        //SPSTAT.7
#define   WCOL          0x40        //SPSTAT.6
sfr       SPCTL   =     0xce;       //SPI控制寄存器
#define   SSIG          0x80        //SPCTL.7
#define   SPEN          0x40        //SPCTL.6
#define   DORD          0x20        //SPCTL.5
#define   MSTR          0x10        //SPCTL.4
#define   CPOL          0x08        //SPCTL.3
```

STC MCU Limited

```
#define   CPHA          0x04                    //SPCTL.2
#define   SPDHH         0x00                    //CPU_CLK/4
#define   SPDH          0x01                    //CPU_CLK/16
#define   SPDL          0x02                    //CPU_CLK/64
#define   SPDLL         0x03                    //CPU_CLK/128
sfr       SPDAT   =     0xcf;                   //SPI数据寄存器
sbit      SPISS   =     P1^1;                   //SPI从机选择口, 连接到其它MCU的SS口
                                                //当SPI为一主多从模式时,
                                                //请使用主机的普通IO口连接到从机的SS口

sfr       IE2     =     0xAF;                   //中断控制寄存器2
#define   ESPI          0x02                    //IE2.1

void InitUart();
void InitSPI();
void SendUart(BYTE dat);                        //发送数据到PC
BYTE RecvUart();                                //从PC接收数据

////////////////////////////////////////////////////////

void main()
{
        InitUart();                             //初始化串口
        InitSPI();                              //初始化SPI
        IE2 |= ESPI;
        EA = 1;

        while (1)
        {
        #ifdef  MASTER                          //对于主机(接收串口数据 并发送给从机,同时
                                                // 从即接收SPI数据并回传给PC)
                ACC = RecvUart();
                SPISS = 0;                      //拉低从机的SS
                SPDAT = ACC;                    //触发SPI发送数据
        #endif
        }
}

////////////////////////////////////////////////////////
void spi_isr() interrupt 9 using 1              //SPI中断服务程序 9 (004BH)
{
        SPSTAT = SPIF | WCOL;                   //清除SPI状态位
#ifdef MASTER
        SPISS = 1;                              //拉高从机的SS
        SendUart(SPDAT);                        //返回SPI数据
#else                                           //对于从机(从主机接收SPI数据,同时
        SPDAT = SPDAT;                          //发送前一个SPI数据给主机)
#endif
}
```

STC MCU Limited

```
//////////////////////////////////////////////////////////
void InitUart()
{
        SCON    =       0x5a;                   //设置串口为8位可变波特率
#if     URMD    ==      0
        T2L     =       0xd8;                   //设置波特率重装值
        T2H     =       0xff;                   //115200 bps(65536-18432000/4/115200)
        AUXR    =       0x14;                   //T2为1T模式, 并启动定时器2
        AUXR    |=      0x01;                   //选择定时器2为串口1的波特率发生器
#elif   URMD    ==      1
        AUXR    =       0x40;                   //定时器1为1T模式
        TMOD    =       0x00;                   //定时器1为模式0(16位自动重载)
        TL1     =       0xd8;                   //设置波特率重装值
        TH1     =       0xff;                   //115200 bps(65536-18432000/4/115200)
        TR1     =       1;                      //定时器1开始启动
#else
        TMOD    =       0x20;                   //设置定时器1为8位自动重装载模式
        AUXR    =       0x40;                   //定时器1为1T模式
        TH1 = TL1 =     0xfb;                   //115200 bps(256 - 18432000/32/115200)
        TR1     =       1;                      //定时器1开始启动
#endif
}

//////////////////////////////////////////////////////////
void InitSPI()
{
        SPDAT = 0;                              //初始化SPI数据
        SPSTAT = SPIF | WCOL;                   //清除SPI状态位
#ifdef MASTER
        SPCTL = SPEN | MSTR;                    //主机模式
#else
        SPCTL = SPEN;                           //从机模式
#endif
}

//////////////////////////////////////////////////////////
void SendUart(BYTE dat)
{
        while (!TI);                            //等待发送完成
        TI = 0;                                 //清除发送标志
        SBUF = dat;                             //发送串口数据
}

//////////////////////////////////////////////////////////
BYTE RecvUart()
{
        while (!RI);                            //等待串口数据接收完成
        RI = 0;                                 //清除接收标志
        return SBUF;                            //返回串口数据
}
```

STC MCU Limited

2. 汇编程序

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ---------------------------------------------------------*/
/* --- 演示STC 1T 系列单片机  SPI功能(适用单主单从，中断方式)----------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------------------------------------------------------- */
/* --- Web: www.STCMCU.com ------------------------------------------------------------ */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序---------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序---------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#define   MASTER                //define:master undefine:slave

#define   URMD    0             //0:使用定时器2作为波特率发生器
                                //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
T2H       DATA    0D6H          //定时器2高8位
T2L       DATA    0D7H          //定时器2低8位

AUXR      DATA    08EH          ;辅助寄存器
SPSTAT    DATA    0CDH          ;SPI状态寄存器
SPIF      EQU     080H          ;SPSTAT.7
WCOL      EQU     040H          ;SPSTAT.6
SPCTL     DATA    0CEH          ;SPI控制寄存器
SSIG      EQU     080H          ;SPCTL.7
SPEN      EQU     040H          ;SPCTL.6
DORD      EQU     020H          ;SPCTL.5
MSTR      EQU     010H          ;SPCTL.4
CPOL      EQU     008H          ;SPCTL.3
CPHA      EQU     004H          ;SPCTL.2
SPDHH     EQU     000H          ;CPU_CLK/4
SPDH      EQU     001H          ;CPU_CLK/16
SPDL      EQU     002H          ;CPU_CLK/64
SPDLL     EQU     003H          ;CPU_CLK/128
SPDAT     DATA    0CFH          ;SPI数据寄存器
SPISS     BIT     P1.1          ;SPI从机选择口, 连接到其它MCU的SS口
                                ;当SPI为一主多从模式时,请使用主机的普通IO口连接到从机的SS口


IE2       EQU     0AFH          ;中断控制寄存器2
ESPI      EQU     02H           ;IE2.1

;//////////////////////////////////////////////////////////
```

STC MCU Limited

```
        ORG     0000H
        LJMP    RESET
        ORG     004BH                           ;SPI中断服务程序
SPI_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     SPSTAT, #SPIF | WCOL            ;清除SPI状态位
#ifdef MASTER
        SETB    SPISS                           ;拉高从机的SS
        MOV     A,      SPDAT                   ;返回SPI数据
        LCALL   SEND_UART
#else                                           //对于从机(从主机接收SPI数据,同时
        MOV     SPDAT,  SPDAT                   ;发送前一个SPI数据给主机)
#endif
        POP     PSW
        POP     ACC
        RETI

;/////////////////////////////////////////////////////////

        ORG     0100H
RESET:
        LCALL   INIT_UART                       ;初始化串口
        LCALL   INIT_SPI                        ;初始化SPI
        ORL     IE2,    #ESPI
        SETB    EA
MAIN:
#ifdef MASTER                                   //对于主机(接收串口数据 并发送给从机,同时
        LCALL   RECV_UART                       ; 从从即接收SPI数据并回传给PC)
        CLR     SPISS                           ;拉低从机的SS
        MOV     SPDAT,  A                       ;触发SPI发送数据
#endif
        SJMP    MAIN

;/////////////////////////////////////////////////////////

INIT_UART:
        MOV     SCON,   #5AH                    ;设置串口为8位可变波特率
#if URMD == 0
        MOV     T2L,    #0D8H                   ;设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H                    ;T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H                    ;选择定时器2为串口1的波特率发生器
#elif URMD == 1
        MOV     AUXR,   #40H                    ;定时器1为1T模式
        MOV     TMOD,   #00H                    ;定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H                   ;设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                             ;定时器1开始运行
```

STC MCU Limited

```
#else
        MOV     TMOD,   #20H                    ;设置定时器1为8位自动重装载模式
        MOV     AUXR,   #40H                    ;定时器1为1T模式
        MOV     TL1,    #0FBH                   ;115200 bps(256 - 18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1
#endif
        RET

;//////////////////////////////////////////////////////////

INIT_SPI:
        MOV     SPDAT,  #0                      ;初始化SPI数据
        MOV     SPSTAT, #SPIF | WCOL            ;清除SPI状态位
#ifdef MASTER
        MOV     SPCTL,  #SPEN | MSTR            ;主机模式
#else
        MOV     SPCTL,  #SPEN                   ;从机模式
#endif
        RET

;//////////////////////////////////////////////////////////

SEND_UART:
        JNB     TI,     $                       ;等待发送完成
        CLR     TI                              ;清除发送标志
        MOV     SBUF,   A                       ;发送串口数据
        RET

;//////////////////////////////////////////////////////////

RECV_UART:
        JNB     RI,     $                       ;等待串口数据接收完成
        CLR     RI                              ;清除接收标志
        MOV     A,      SBUF                    ;返回串口数据
        RET
        RET

;//////////////////////////////////////////////////////////

        END
```

STC MCU Limited

14.4.2 查询方式

1. C程序

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------------*/
/* --- 演示STC 1T 系列单片机 SPI功能(适用单主单从，查询方式)----------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------- ---------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------- ---------------------------------------------------------- */
/* --- Web: www.STCMCU.com ---------- ------------------------------------------------------------ */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序---------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序---------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

//#define  MASTER                                          //define:master undefine:slave
#define   FOSC      18432000L
#define   BAUD      (256 - FOSC / 32 / 115200)

typedef   unsigned char      BYTE;
typedef   unsigned int       WORD;
typedef   unsigned long      DWORD;

#define    URMD   0                  //0:使用定时器2作为波特率发生器
                                     //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                     //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器

sfr       T2H   = 0xd6;                    //定时器2高8位
sfr       T2L   = 0xd7;                    //定时器2低8位

sfr       AUXR    =       0x8e;            //辅助寄存器
sfr       SPSTAT  =       0xcd;            //SPI状态寄存器
#define   SPIF            0x80             //SPSTAT.7
#define   WCOL            0x40             //SPSTAT.6
sfr       SPCTL   =       0xce;            //SPI控制寄存器
#define   SSIG            0x80             //SPCTL.7
#define   SPEN            0x40             //SPCTL.6
#define   DORD            0x20             //SPCTL.5
#define   MSTR            0x10             //SPCTL.4
#define   CPOL            0x08             //SPCTL.3
#define   CPHA            0x04             //SPCTL.2
#define   SPDHH           0x00             //CPU_CLK/4
#define   SPDH            0x01             //CPU_CLK/16
#define   SPDL            0x02             //CPU_CLK/64
```

STC MCU Limited

```
#define   SPDLL          0x03                    //CPU_CLK/128
sfr       SPDAT   =      0xcf;                   //SPI数据寄存器
sbit      SPISS   =      P1^1;                   //SPI从机选择口, 连接到其它MCU的SS口
                                                 //当SPI为一主多从模式时,
                                                 //请使用主机的普通IO口连接到从机的SS口

void InitUart();
void InitSPI();
void SendUart(BYTE dat);                         //发送数据到PC
BYTE RecvUart();                                 //从PC接收数据
BYTE SPISwap(BYTE dat);                          //主机与从机之间交换数据
//////////////////////////////////////////////////////////
void main()
{
        InitUart();                              //初始化串口
        InitSPI();                               //初始化SPI

        while (1)
        {
        #ifdef MASTER                            //对于主机(接收串口数据 并发送给从机,同时
                                                 //        从从即接收SPI数据并回传给PC)
                SendUart(SPISwap(RecvUart()));
        #else                                    //对于从机(从主机接收SPI数据,同时
                ACC = SPISwap(ACC);              //        发送前一个SPI数据给主机)
        #endif
        }
}
//////////////////////////////////////////////////////////
void InitUart()
{
        SCON    =      0x5a;                     //设置串口为8位可变波特率
#if     URMD    ==     0
        T2L     =      0xd8;                     //设置波特率重装值
        T2H     =      0xff;                     //115200 bps(65536-18432000/4/115200)
        AUXR    =      0x14;                     //T2为1T模式, 并启动定时器2
        AUXR    |=     0x01;                     //选择定时器2为串口1的波特率发生器
#elif   URMD    ==     1
        AUXR    =      0x40;                     //定时器1为1T模式
        TMOD    =      0x00;                     //定时器1为模式0(16位自动重载)
        TL1     =      0xd8;                     //设置波特率重装值
        TH1     =      0xff;                     //115200 bps(65536-18432000/4/115200)
        TR1     =      1;                        //定时器1开始启动
#else
        TMOD    =      0x20;                     //设置定时器1为8位自动重装载模式
        AUXR    =      0x40;                     //定时器1为1T模式
        TH1 =  TL1 =   0xfb;                     //115200 bps(256 - 18432000/32/115200)
        TR1     =      1;
#endif
}
```

STC MCU Limited

```
//////////////////////////////////////////////////////////

void InitSPI()
{
        SPDAT = 0;                          //初始化SPI数据
        SPSTAT = SPIF | WCOL;               //清除SPI状态位
#ifdef MASTER
        SPCTL = SPEN | MSTR;                //主机模式
#else
        SPCTL = SPEN;                       //从机模式
#endif
}

//////////////////////////////////////////////////////////

void SendUart(BYTE dat)
{
        while (!TI);                        //等待发送完成
        TI = 0;                             //清除发送标志
        SBUF = dat;                         //发送串口数据
}

//////////////////////////////////////////////////////////

BYTE RecvUart()
{
        while (!RI);                        //等待串口数据接收完成
        RI = 0;                             //清除接收标志
        return SBUF;                        //返回串口数据
}

//////////////////////////////////////////////////////////

BYTE SPISwap(BYTE dat)
{
#ifdef MASTER
        SPISS = 0;                          //拉低从机的SS
#endif
        SPDAT = dat;                        //触发SPI发送数据
        while (!(SPSTAT & SPIF));           //等待发送完成
        SPSTAT = SPIF | WCOL;               //清除SPI状态位
#ifdef MASTER
        SPISS = 1;                          //拉高从机的SS
#endif
        return SPDAT;                       //返回SPI数据
}
```

STC MCU Limited

2. 汇编程序

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ---------------------------------------------------------------*/
/* --- 演示STC 1T 系列单片机  SPI功能(适用单主单从，查询方式)---------------------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ----------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序---------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序---------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

//#define  MASTER                 //define:master undefine:slave

#define   URMD   0                //0:使用定时器2作为波特率发生器
                                  //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                  //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
T2H       DATA   0D6H             //定时器2高8位
T2L       DATA   0D7H             //定时器2低8位

AUXR      DATA   08EH                   ;Auxiliary register
SPSTAT    DATA   0CDH                   ;SPI status register
SPIF      EQU    080H                   ;SPSTAT.7
WCOL      EQU    040H                   ;SPSTAT.6
SPCTL     DATA   0CEH                   ;SPI control register
SSIG      EQU    080H                   ;SPCTL.7
SPEN      EQU    040H                   ;SPCTL.6
DORD      EQU    020H                   ;SPCTL.5
MSTR      EQU    010H                   ;SPCTL.4
CPOL      EQU    008H                   ;SPCTL.3
CPHA      EQU    004H                   ;SPCTL.2
SPDHH     EQU    000H                   ;CPU_CLK/4
SPDH      EQU    001H                   ;CPU_CLK/16
SPDL      EQU    002H                   ;CPU_CLK/64
SPDLL     EQU    003H                   ;CPU_CLK/128
SPDAT     DATA   0CFH                   ;SPI data register
SPISS     BIT    P1.1                   ;SPI从机选择口, 连接到其它MCU的SS口
                                        ;当SPI为一主多从模式时,
                                        ;请使用主机的普通IO口连接到从机的SS口

;///////////////////////////////////////////////////////////
```

STC MCU Limited

```
        ORG     0000H
        LJMP    RESET
        ORG     0100H
RESET:
        LCALL   INIT_UART               ;初始化串口
        LCALL   INIT_SPI                ;初始化SPI
MAIN:
#ifdef MASTER                           //对于主机(接收串口数据 并发送给从机,同时
        LCALL   RECV_UART               ;       从从即接收SPI数据并回传给PC)
        LCALL   SPI_SWAP
        LCALL   SEND_UART
#else                                   //对于从机(从主机接收SPI数据,同时
        LCALL   SPI_SWAP                ;       发送前一个SPI数据给主机)
#endif
        SJMP    MAIN

;///////////////////////////////////////////////////////

INIT_UART:
        MOV     SCON,   #5AH            ;设置串口为8位可变波特率
#if URMD == 0
        MOV     T2L,    #0D8H           ;设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H            ;T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H            ;选择定时器2为串口1的波特率发生器
#elif URMD == 1
        MOV     AUXR,   #40H            ;定时器1为1T模式
        MOV     TMOD,   #00H            ;定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H           ;设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                     ;定时器1开始运行
#else
        MOV     TMOD,   #20H            ;设置定时器1为8位自动重装载模式
        MOV     AUXR,   #40H            ;定时器1为1T模式
        MOV     TL1,    #0FBH           ;115200 bps(256 - 18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1
#endif
        RET

;///////////////////////////////////////////////////////

INIT_SPI:
        MOV     SPDAT,  #0                      ;初始化SPI数据
        MOV     SPSTAT, #SPIF | WCOL            ;清除SPI状态位
#ifdef MASTER
        MOV     SPCTL,  #SPEN | MSTR            ;主机模式
#else
```

STC MCU Limited

```
        MOV     SPCTL,  #SPEN                   ;从机模式
#endif
        RET

;/////////////////////////////////////////////////////////

SEND_UART:
        JNB     TI,     $                       ;等待发送完成
        CLR     TI                              ;清除发送标志
        MOV     SBUF,   A                       ;发送串口数据
        RET

;/////////////////////////////////////////////////////////

RECV_UART:
        JNB     RI,     $                       ;等待串口数据接收完成
        CLR     RI                              ;清除接收标志
        MOV     A,      SBUF                    ;返回串口数据
        RET
        RET

;/////////////////////////////////////////////////////////

SPI_SWAP:
#ifdef MASTER
        CLR     SPISS                           ;拉低从机的SS
#endif
        MOV     SPDAT,  A                       ;触发SPI发送数据
WAIT:
        MOV     A,      SPSTAT
        JNB     ACC.7,  WAIT                    ;等待发送完成
        MOV     SPSTAT, #SPIF | WCOL            ;清除SPI状态位
#ifdef MASTER
        SETB    SPISS                           ;拉高从机的SS
#endif
        MOV     A,      SPDAT                   ;返回SPI数据
        RET

;/////////////////////////////////////////////////////////

        END
```

STC MCU Limited

14.5 适用互为主从系统的SPI功能测试程序(C和汇编)

14.5.1 中断方式

1. C程序

```
/*-----------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. --------------- ----------------------------------------------------*/
/* --- 演示STC 1T 系列单片机  SPI功能(适用互为主从系统，中断方式)---------------*/
/* --- 研发顾问QQ：800003751-------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序--------------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序--------------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可------------*/
/*-----------------------------------------------------------------------------------------------*/

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

#define FOSC      18432000L
#define BAUD      (256 - FOSC / 32 / 115200)

typedef unsigned char      BYTE;
typedef unsigned int       WORD;
typedef unsigned long      DWORD;

#define   URMD   0                  //0:使用定时器2作为波特率发生器
                                    //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                    //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器

sfr       T2H     =     0xd6;          //定时器2高8位
sfr       T2L     =     0xd7;          //定时器2低8位

sfr       AUXR    =     0x8e;          //辅助寄存器
sfr       SPSTAT  =     0xcd;          //SPI状态寄存器
#define   SPIF          0x80           //SPSTAT.7
#define   WCOL          0x40           //SPSTAT.6
sfr       SPCTL   =     0xce;          //SPI控制寄存器
#define   SSIG          0x80           //SPCTL.7
#define   SPEN          0x40           //SPCTL.6
#define   DORD          0x20           //SPCTL.5
#define   MSTR          0x10           //SPCTL.4
#define   CPOL          0x08           //SPCTL.3
#define   CPHA          0x04           //SPCTL.2
#define   SPDHH         0x00           //CPU_CLK/4
```

STC MCU Limited

```
#define   SPDH          0x01                //CPU_CLK/16
#define   SPDL          0x02                //CPU_CLK/64
#define   SPDLL         0x03                //CPU_CLK/128
sfr       SPDAT   =     0xcf;               //SPI数据寄存器
sbit      SPISS   =     P1^1;               //SPI从机选择口, 连接到其它MCU的SS口
                                            //当SPI为一主多从模式时,
                                            //请使用主机的普通IO口连接到从机的SS口

sfr       IE2     =     0xAF;               //中断控制寄存器2
#define   ESPI          0x02                //IE2.1

void InitUart();
void InitSPI();
void SendUart(BYTE dat);                    //发送数据到PC
BYTE RecvUart();                            //从PC接收数据

bit       MSSEL;                            //1: master 0:slave

//////////////////////////////////////////////////////////

void main()
{
        InitUart();                         //初始化串口
        InitSPI();                          //初始化SPI
        IE2 |= ESPI;
        EA = 1;

        while (1)
        {
                if (RI)
                {
                        SPCTL = SPEN | MSTR;    //设置为主机模式
                        MSSEL = 1;
                        ACC = RecvUart();
                        SPISS = 0;              //拉低从机的SS
                        SPDAT = ACC;            //触发SPI发送数据
                }
        }
}

//////////////////////////////////////////////////////////

void spi_isr() interrupt 9 using 1          //SPI中断服务程序 9 (004BH)
{
        SPSTAT = SPIF | WCOL;               //清除SPI状态位
        if (MSSEL)
        {
                SPCTL = SPEN;               //重置为从机模式
                MSSEL = 0;
```

STC MCU Limited

```
                SPISS = 1;                      //拉高从机的SS
                SendUart(SPDAT);                //返回SPI数据
        }
        else
        {                                       //对于从机(从主机接收SPI数据,同时
                SPDAT = SPDAT;                  //发送前一个SPI数据给主机)
        }
}

//////////////////////////////////////////////////////////

void InitUart()
{
        SCON    =       0x5a;                   //设置串口为8位可变波特率
#if     URMD    ==      0
        T2L     =       0xd8;                   //设置波特率重装值
        T2H     =       0xff;                   //115200 bps(65536-18432000/4/115200)
        AUXR    =       0x14;                   //T2为1T模式, 并启动定时器2
        AUXR    |=      0x01;                   //选择定时器2为串口1的波特率发生器
#elif   URMD    ==      1
        AUXR    =       0x40;                   //定时器1为1T模式
        TMOD    =       0x00;                   //定时器1为模式0(16位自动重载)
        TL1     =       0xd8;                   //设置波特率重装值
        TH1     =       0xff;                   //115200 bps(65536-18432000/4/115200)
        TR1     =       1;                      //定时器1开始启动
#else
        TMOD    =       0x20;                   //设置定时器1为8位自动重装载模式
        AUXR    =       0x40;                   //定时器1为1T模式
        TH1 =   TL1 =   0xfb;                   //115200 bps(256 - 18432000/32/115200)
        TR1     =       1;
#endif
}

//////////////////////////////////////////////////////////

void InitSPI()
{
        SPDAT = 0;                              //初始化SPI数据
        SPSTAT = SPIF | WCOL;                   //清除SPI状态位
        SPCTL = SPEN;                           //从机模式
}

//////////////////////////////////////////////////////////

void SendUart(BYTE dat)
{
        while (!TI);                            //等待发送完成
        TI = 0;                                 //清除发送标志
        SBUF = dat;                             //发送串口数据
}
```

STC MCU Limited

```
//////////////////////////////////////////////////////////
BYTE RecvUart()
{
        while (!RI);                    //等待串口数据接收完成
        RI = 0;                         //清除接收标志
        return SBUF;                    //返回串口数据
}
```

2. 汇编程序

```
/*-------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ---------------------------------------------------------*/
/* --- 演示STC 1T 系列单片机  SPI功能(适用互为主从系统，中断方式)----------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------- ---------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------- ---------------------------------------------------- */
/* --- Web: www.STCMCU.com ---------- ---------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序---------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序---------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*-------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#define   URMD   0          //0:使用定时器2作为波特率发生器
                            //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                            //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
T2H       DATA   0D6H       //定时器2高8位
T2L       DATA   0D7H       //定时器2低8位

AUXR      DATA   08EH       ;辅助寄存器
SPSTAT    DATA   0CDH       ;SPI状态寄存器
SPIF      EQU    080H       ;SPSTAT.7
WCOL      EQU    040H       ;SPSTAT.6
SPCTL     DATA   0CEH       ;SPI控制寄存器
SSIG      EQU    080H       ;SPCTL.7
SPEN      EQU    040H       ;SPCTL.6
DORD      EQU    020H       ;SPCTL.5
MSTR      EQU    010H       ;SPCTL.4
CPOL      EQU    008H       ;SPCTL.3
CPHA      EQU    004H       ;SPCTL.2
SPDHH     EQU    000H       ;CPU_CLK/4
SPDH      EQU    001H       ;CPU_CLK/16
SPDL      EQU    002H       ;CPU_CLK/64
SPDLL     EQU    003H       ;CPU_CLK/128
SPDAT     DATA   0CFH       ;SPI数据寄存器
SPISS     BIT    P1.1       ;SPI从机选择口, 连接到其它MCU的SS口
                            ;当SPI为一主多从模式时,请使用主机的普通IO口连接到从机的SS口
```

STC MCU Limited

```
IE2     EQU     0AFH                    ;中断控制寄存器2
ESPI    EQU     02H                     ;IE2.1

MSSEL   BIT     20H.0                   ;1: master 0:slave

;/////////////////////////////////////////////////////////

        ORG     0000H
        LJMP    RESET

        ORG     004BH                   ;SPI中断服务程序
SPI_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     SPSTAT, #SPIF | WCOL    ;清除SPI状态位
        JBC     MSSEL,  MASTER_SEND
SLAVE_RECV:
                                        ;对于从机(从主机接收SPI数据,同时
        MOV     SPDAT,  SPDAT           ;       发送前一个SPI数据给主机)
        JMP     SPI_EXIT
MASTER_SEND:
        SETB    SPISS                   ;拉高从机的SS
        MOV     SPCTL,  #SPEN           ;重置为从机模式
        MOV     A,      SPDAT                   ;返回SPI数据
        LCALL   SEND_UART
SPI_EXIT:
        POP     PSW
        POP     ACC
        RETI

;/////////////////////////////////////////////////////////

        ORG     0100H
RESET:
        MOV     SP,     #3FH
        LCALL   INIT_UART               ;初始化串口
        LCALL   INIT_SPI                ;初始化SPI
        ORL     IE2,    #ESPI
        SETB    EA
MAIN:
        JNB     RI,     $               ;等待串口数据
        MOV     SPCTL,  #SPEN | MSTR    ; ;设置为主机模式
        SETB    MSSEL
        LCALL   RECV_UART               ;接收串口数据
        CLR     SPISS                   ;拉低从机的SS
        MOV     SPDAT,  A               ;触发SPI发送数据
        SJMP    MAIN
```

STC MCU Limited

```
;/////////////////////////////////////////////////////////
INIT_UART:
        MOV     SCON,   #5AH                    ;设置串口为8位可变波特率
#if URMD == 0
        MOV     T2L,    #0D8H                   ;设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H                    ;T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H                    ;选择定时器2为串口1的波特率发生器
#elif URMD == 1
        MOV     AUXR,   #40H                    ;定时器1为1T模式
        MOV     TMOD,   #00H                    ;定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H                   ;设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                             ;定时器1开始运行
#else
        MOV     TMOD,   #20H                    ;设置定时器1为8位自动重装载模式
        MOV     AUXR,   #40H                    ;定时器1为1T模式
        MOV     TL1,    #0FBH                   ;115200 bps(256 - 18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1
#endif
        RET
;/////////////////////////////////////////////////////////

INIT_SPI:
        MOV     SPDAT,  #0                      ;初始化SPI数据
        MOV     SPSTAT, #SPIF | WCOL            ;清除SPI状态位
        MOV     SPCTL,  #SPEN                   ;从机模式
        RET
;/////////////////////////////////////////////////////////

SEND_UART:
        JNB     TI,     $                       ;等待发送完成
        CLR     TI                              ;清除发送标志
        MOV     SBUF,   A                       ;发送串口数据
        RET

;/////////////////////////////////////////////////////////

RECV_UART:
        JNB     RI,     $                       ;等待串口数据接收完成
        CLR     RI                              ;清除接收标志
        MOV     A,      SBUF                    ;返回串口数据
        RET
        RET

;/////////////////////////////////////////////////////////
        END
```

STC MCU Limited

14.5.2 查询方式

1. C程序

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ----------------------------------------------------------*/
/* --- 演示STC 1T 系列单片机  SPI功能(适用互为主从系统，查询方式)----------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------- ---------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------- ---------------------------------------------------- */
/* --- Web: www.STCMCU.com ---------- ---------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序---------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序---------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#include "reg51.h"

#define   FOSC      18432000L
#define   BAUD      (256 - FOSC / 32 / 115200)

typedef unsigned char       BYTE;
typedef unsigned int        WORD;
typedef unsigned long       DWORD;

#define   URMD  0           //0:使用定时器2作为波特率发生器
                            //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                            //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器

sfr       T2H     =         0xd6;           //定时器2高8位
sfr       T2L     =         0xd7;           //定时器2低8位

sfr       AUXR    =         0x8e;           //辅助寄存器
sfr       SPSTAT  =         0xcd;           //SPI状态寄存器
#define   SPIF              0x80            //SPSTAT.7
#define   WCOL              0x40            //SPSTAT.6
sfr       SPCTL   =         0xce;           //SPI控制寄存器
#define   SSIG              0x80            //SPCTL.7
#define   SPEN              0x40            //SPCTL.6
#define   DORD              0x20            //SPCTL.5
#define   MSTR              0x10            //SPCTL.4
#define   CPOL              0x08            //SPCTL.3
#define   CPHA              0x04            //SPCTL.2
#define   SPDHH             0x00            //CPU_CLK/4
#define   SPDH              0x01            //CPU_CLK/16
#define   SPDL              0x02            //CPU_CLK/64
#define   SPDLL             0x03            //CPU_CLK/128
```

STC MCU Limited

```
sfr     SPDAT   =       0xcf;                   //SPI数据寄存器
sbit    SPISS   =       P1^1;                   //SPI从机选择口, 连接到其它MCU的SS口
                                                //当SPI为一主多从模式时,
                                                //请使用主机的普通IO口连接到从机的SS口

void InitUart();
void InitSPI();
void SendUart(BYTE dat);                        //发送数据到PC
BYTE RecvUart();                                //从PC接收数据
BYTE SPISwap(BYTE dat);                         //主机与从机之间交换数据

//////////////////////////////////////////////////////////

void main()
{
        InitUart();                             //初始化串口
        InitSPI();                              //初始化SPI

        while (1)
        {
                if (RI)
                {
                        SPCTL = SPEN | MSTR;            //设置为主机模式
                        SendUart(SPISwap(RecvUart()));
                        SPCTL = SPEN;                   //重置为从机模式
                }
                if (SPSTAT & SPIF)
                {
                        SPSTAT = SPIF | WCOL;           //清除SPI状态位
                        SPDAT = SPDAT;                  //数据从接收缓冲区移到发送缓冲区
                }
        }
}

//////////////////////////////////////////////////////////

void InitUart()
{
        SCON    =       0x5a;                   //设置串口为8位可变波特率
#if     URMD    ==      0
        T2L     =       0xd8;                   //设置波特率重装值
        T2H     =       0xff;                   //115200 bps(65536-18432000/4/115200)
        AUXR    =       0x14;                   //T2为1T模式, 并启动定时器2
        AUXR    |=      0x01;                   //选择定时器2为串口1的波特率发生器
#elif   URMD    ==      1
        AUXR    =       0x40;                   //定时器1为1T模式
        TMOD    =       0x00;                   //定时器1为模式0(16位自动重载)
        TL1     =       0xd8;                   //设置波特率重装值
        TH1     =       0xff;                   //115200 bps(65536-18432000/4/115200)
```

STC MCU Limited

```
        TR1     =       1;                              //定时器1开始启动
#else
        TMOD    =       0x20;                   //设置定时器1为8位自动重装载模式
        AUXR    =       0x40;                   //定时器1为1T模式
        TH1 =   TL1 =   0xfb;                   //115200 bps(256 - 18432000/32/115200)
        TR1     =       1;
#endif
}

//////////////////////////////////////////////////////////

void InitSPI()
{
        SPDAT = 0;                              //初始化SPI数据
        SPSTAT = SPIF | WCOL;                   //清除SPI状态位
        SPCTL = SPEN;                           //从机模式
}

//////////////////////////////////////////////////////////

void SendUart(BYTE dat)
{
        while (!TI);                            //等待发送完成
        TI = 0;                                 //清除发送标志
        SBUF = dat;                             //发送串口数据
}

//////////////////////////////////////////////////////////

BYTE RecvUart()
{
        while (!RI);                            //等待串口数据接收完成
        RI = 0;                                 //清除接收标志
        return SBUF;                            //返回串口数据
}

//////////////////////////////////////////////////////////

BYTE SPISwap(BYTE dat)
{
        SPISS = 0;                              //拉低从机的SS
        SPDAT = dat;                            //触发SPI发送数据
        while (!(SPSTAT & SPIF));               //等待发送完成
        SPSTAT = SPIF | WCOL;                   //清除SPI状态位
        SPISS = 1;                              //拉高从机的SS
        return SPDAT;                           //返回SPI数据
}
```

STC MCU Limited

2. 汇编程序

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited. ---------- ---------------------------------------------------------------*/
/* --- 演示STC 1T 系列单片机  SPI功能(适用互为主从系统，查询方式)----------*/
/* --- 研发顾问QQ：800003751----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ---------------------------------------------------------------------- */
/* --- Tel: 86-755-82948412 ---------------------------------------------------------------------- */
/* --- Web: www.STCMCU.com ----------------------------------------------------------------------- */
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序---------- */
/* 如果要在文章中应用此代码,请在文章中注明使用了STC的资料及程序---------- */
/*---- 在 Keil C 开发环境中选择 Intel 8052编译，头文件包含<reg51.h>即可----------*/
/*---------------------------------------------------------------------------------------------------- */

//假定测试芯片的工作频率为18.432MHz

#define   URMD   0              //0:使用定时器2作为波特率发生器
                                //1:使用定时器1的模式0(16位自动重载模式)作为波特率发生器
                                //2:使用定时器1的模式2(8位自动重载模式)作为波特率发生器
T2H       DATA   0D6H           //定时器2高8位
T2L       DATA   0D7H           //定时器2低8位

AUXR      DATA   08EH           ;辅助寄存器
SPSTAT    DATA   0CDH           ;SPI状态寄存器
SPIF      EQU    080H           ;SPSTAT.7
WCOL      EQU    040H           ;SPSTAT.6
SPCTL     DATA   0CEH           ;SPI控制寄存器
SSIG      EQU    080H           ;SPCTL.7
SPEN      EQU    040H           ;SPCTL.6
DORD      EQU    020H           ;SPCTL.5
MSTR      EQU    010H           ;SPCTL.4
CPOL      EQU    008H           ;SPCTL.3
CPHA      EQU    004H           ;SPCTL.2
SPDHH     EQU    000H           ;CPU_CLK/4
SPDH      EQU    001H           ;CPU_CLK/16
SPDL      EQU    002H           ;CPU_CLK/64
SPDLL     EQU    003H           ;CPU_CLK/128
SPDAT     DATA   0CFH           ;SPI数据寄存器
SPISS     BIT    P1.1           ;SPI从机选择口, 连接到其它MCU的SS口
                                ;当SPI为一主多从模式时,请使用主机的普通IO口连接到从机的SS口

;///////////////////////////////////////////////////////

          ORG    0000H
          LJMP   RESET
          ORG    0100H
RESET:
          LCALL  INIT_UART      ;初始化串口
```

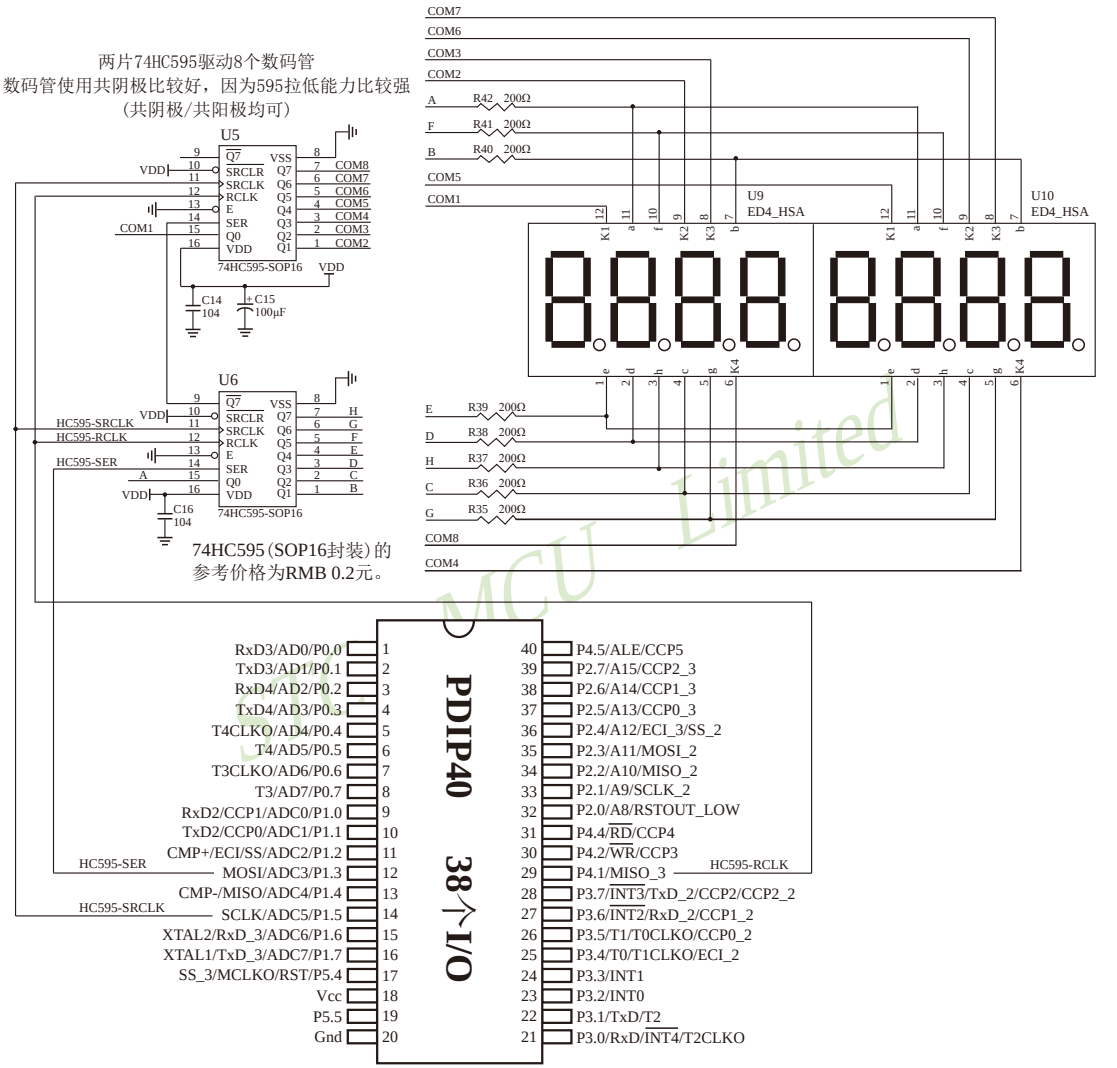
```
        LCALL   INIT_SPI            ;初始化SPI
MAIN:
        JB      RI,     MASTER_MODE
SLAVE_MODE:
        MOV     A,      SPSTAT
        JNB     ACC.7,  MAIN
        MOV     SPSTAT, #SPIF | WCOL        ;清除SPI状态位
        MOV     SPDAT,  SPDAT               ;返回SPI数据
        SJMP    MAIN
MASTER_MODE:
        MOV     SPCTL,  #SPEN | MSTR        ;设置为主机模式
        LCALL   RECV_UART                   ;接收串口数据
        LCALL   SPI_SWAP                    ;发送串口数据给从机,同时从从机接收SPI数据
        LCALL   SEND_UART                   ;发送SPI数据到PC
        MOV     SPCTL,  #SPEN;              ;重置为从机模式
        SJMP    MAIN

;//////////////////////////////////////////////////////////

INIT_UART:
        MOV     SCON,   #5AH                ;设置串口为8位可变波特率
#if URMD == 0
        MOV     T2L,    #0D8H               ;设置波特率重装值(65536-18432000/4/115200)
        MOV     T2H,    #0FFH
        MOV     AUXR,   #14H                ;T2为1T模式, 并启动定时器2
        ORL     AUXR,   #01H                ;选择定时器2为串口1的波特率发生器
#elif URMD == 1
        MOV     AUXR,   #40H                ;定时器1为1T模式
        MOV     TMOD,   #00H                ;定时器1为模式0(16位自动重载)
        MOV     TL1,    #0D8H               ;设置波特率重装值(65536-18432000/4/115200)
        MOV     TH1,    #0FFH
        SETB    TR1                         ;定时器1开始运行
#else
        MOV     TMOD,   #20H                ;设置定时器1为8位自动重装载模式
        MOV     AUXR,   #40H                ;定时器1为1T模式
        MOV     TL1,    #0FBH               ;115200 bps(256 - 18432000/32/115200)
        MOV     TH1,    #0FBH
        SETB    TR1
#endif
        RET

;//////////////////////////////////////////////////////////

INIT_SPI:
        MOV     SPDAT,  #0                  ;初始化SPI数据
        MOV     SPSTAT, #SPIF | WCOL        ;清除SPI状态位
        MOV     SPCTL,  #SPEN               ;从机模式
        RET
```

STC MCU Limited

```
;/////////////////////////////////////////////////////////
SEND_UART:
        JNB     TI,     $               ;等待发送完成
        CLR     TI                      ;清除发送标志
        MOV     SBUF,   A               ;发送串口数据
        RET

;/////////////////////////////////////////////////////////

RECV_UART:
        JNB     RI,     $               ;等待串口数据接收完成
        CLR     RI                      ;清除接收标志
        MOV     A,      SBUF            ;返回串口数据
        RET
        RET

;/////////////////////////////////////////////////////////

SPI_SWAP:
        CLR     SPISS                   ;拉低从机的SS
        MOV     SPDAT,  A               ;触发SPI发送数据
WAIT:
        MOV     A,      SPSTAT
        JNB     ACC.7,  WAIT            ;等待发送完成
        MOV     SPSTAT, #SPIF | WCOL    ;清除SPI状态位
        SETB    SPISS                   ;拉高从机的SS
        MOV     A,      SPDAT           ;返回SPI数据
        RET

;/////////////////////////////////////////////////////////

        END
```

STC MCU Limited

14.6 利用SPI控制74HC595驱动8位数码管及测试程序(C和汇编)



下面是用STC的MCU的SPI方式控制74HC595驱动8位数码管(串口扩展，3根线)的测试程序：

1. C程序

```
/*---------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------*/
/* --- STC 1T Series MCU Programme Demo ------------------------------*/
/* --- Fax: 86-755-82944243 ------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------*/
/* If you want to use the program or the program referenced in the  */
/* article, please specify in which data and procedures from STC    */
/*---------------------------------------------------------------------*/

/*     本程序经过测试完全正常, 不提供电话技术支持, 如不能理解, 请自行补充相关基础.  */


/*************    本程序功能说明    **************

用STC的MCU的SPI方式控制74HC595驱动8位数码管。

用户可以修改宏来选择时钟频率, 可以修改寄存器定义是STC12C5A60S2系列 还是 STC12C5628AD
STC12C5410AD  STC12C4052AD系列.

用户可以在显示函数里修改成共阴或共阳.推荐尽量使用共阴数码管.

显示效果为: 8个数码管循环显示0,1,2...,A,B..F,消隐.

*****************************************/

#include  "reg52.h"


/***************************** 用户定义宏 ***********************************/

#define   MAIN_Fosc       11059200UL          //定义主时钟
//#define MAIN_Fosc       22118400UL          //定义时钟

/*****************************************************************************/

sfr       SPSTAT  =       0xCD;       //STC12C5A60S2系列
sfr       SPCTL   =       0xCE;       //STC12C5A60S2系列
sfr       SPDAT   =       0xCF;       //STC12C5A60S2系列

/*
sfr       SPSTAT  =       0x84;       //STC12C5628AD  STC12C5410AD  STC12C4052AD系列
sfr       SPCTL   =       0x85;       //STC12C5628AD  STC12C5410AD  STC12C4052AD系列
sfr       SPDAT   =       0x86;       //STC12C5628AD  STC12C5410AD  STC12C4052AD系列
*/
```

STC MCU Limited

```
/********************* 下面的宏自动生成,用户不可修改 ***************************/

#define    Timer0_Reload      (MAIN_Fosc / 12000)
/*****************************************************************************/

//SPCTL             SPI控制寄存器
//  7     6     5     4     3     2     1     0       Reset Value
//SSIG    SPEN    DORD    MSTR    CPOL    CPHA    SPR1    SPR0                0x00
#define   SSIG    1                                 //1: 忽略SS脚，由MSTR位决定主机还是从机
                                                    //0: SS脚用于决定主从机。
#define   SPEN    1                                 //1: 允许SPI，
                                                    //0：禁止SPI，所有SPI管脚均为普通IO
#define   DORD    0                                 //1：LSB先发，
                                                    //0：MSB先发
#define   MSTR    1                                 //1：设为主机
                                                    //0：设为从机
#define   CPOL    1                                 //1: 空闲时SCLK为高电平，
                                                    //0：空闲时SCLK为低电平
#define   CPHA    1                                 //
#define   SPR1    0                                 //SPR1,SPR0   00: fosc/4,    01: fosc/16
#define   SPR0    0                                 //          10: fosc/64,    11: fosc/128
#define   SPEED_4           0                       // fosc/4
#define   SPEED_16          1                       // fosc/16
#define   SPEED_64          2                       // fosc/64
#define   SPEED_128         3                       // fosc/128

//SPSTATSPI状态寄存器
//  7     6     5  4  3  2  1  0           Reset Value
//          SPIF    WCOL    -          -          -          -          -          -
#define   SPIF    0x80                              //SPI传输完成标志。写入1清0。
#define   WCOL    0x40                              //SPI写冲突标志。写入1清0。

/**************    本地常量声明      ***************/
unsigned char code t_display[]={
//          0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F   消隐
        0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,0x00};
//段码

unsigned char code T_COM[]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};               //位码

/**************    本地变量声明      ***************/
sbit        SPI_SCL     =   P1^5;       //SPI同步时钟    P_HC595_SRCLK      pin 11   SRCLK Shift data clock
//sbit      SPI_MISO    =   P1^6;       //SPI同步数据输入    本例不用
sbit        SPI_MOSI    =   P1^3;       //SPI同步数据输出     P_HC595_SER   pin 14   SER     data input
sbit        P_HC595_RCLK = P4^1;        //SPI片选(任意IO)     pin 12          RCLk    store (latch) clock

unsigned char       LED8[8];            //显示缓冲
unsigned char       display_index;                              //显示位索引
bit         B_1ms;                                              //1ms标志
```

STC MCU Limited

```
/*********************************************/
void main(void)                                         //主函数
{
    unsigned char   i,k;
    unsigned int    j;

    SPCTL = (SSIG << 7) + (SPEN << 6) + (DORD << 5) + (MSTR << 4) + (CPOL << 3) + (CPHA << 2)
        + SPEED_4;                                      //配置SPI

    TMOD  =     0x01;                                   //Timer 0 config as 16bit timer, 12T
    TH0   =     (65536 - Timer0_Reload) / 256;
    TL0   =     (65536 - Timer0_Reload) % 256;
    ET0   =     1;
    TR0   =     1;
    EA    =     1;

    for(i=0; i<8; i++)  LED8[i] = 0x10;                 //上电消隐
    j = 0;
    k = 0;

    while(1)
    {
        while(!B_1ms)       ;                           //等待1ms到
        B_1ms = 0;
        if(++j >= 500)                                  //500ms到
        {
            j = 0;
            for(i=0; i<8; i++)  LED8[i] = k;            //刷新显示
            if(++k > 0x10)      k = 0;                  //8个数码管循环显示0,1,2...,A,B..F,消隐.
        }
    }
}
/*********************************************/

/*********************************************/
void SPI_SendByte(unsigned char dat) //SPI发送一个字节
{
    SPSTAT = SPIF + WCOL;                               //清0 SPIF和WCOL标志
    SPDAT = dat;                                        //发送一个字节
    while((SPSTAT & SPIF) == 0)         ;               //等待发送完成
    SPSTAT = SPIF + WCOL;                               //清0 SPIF和WCOL标志
}

/*********************************************/
void DisplayScan(void)                                  //显示扫描函数
{
//  SPI_SendByte(~T_COM[display_index]);                //共阴    输出位码
//  SPI_SendByte(t_display[LED8[display_index]]);       //共阴    输出段码
    SPI_SendByte(T_COM[display_index]);                 //共阳    输出位码
```

STC MCU Limited

```
        SPI_SendByte(~t_display[LED8[display_index]]);          //共阳    输出段码
        P_HC595_RCLK = 1;
        P_HC595_RCLK = 0;                                       //锁存输出数据
        if(++display_index >= 8)      display_index = 0;        //8位结束回0
}


/*********************************************/
void timer0 (void) interrupt 1 //Timer0 1ms中断函数
{
        TH0  = (65536 - Timer0_Reload) / 256;                   //重装定时值
        TL0  = (65536 - Timer0_Reload) % 256;

        DisplayScan();                                          //1ms扫描显示一位
        B_1ms = 1;                                              //1ms标志
}
```

STC MCU Limited

2. 汇编程序

```
;/*---------------------------------------------------------------------------*/
;/* --- STC MCU International Limited ------------------------------------*/
;/* --- STC 1T Series MCU Programme Demo ---------------------------------*/
;/* --- Fax: 86-755-82944243 ---------------------------------------------*/
;/* --- Tel: 86-755-82948412 ---------------------------------------------*/
;/* --- Web: www.STCMCU.com ----------------------------------------------*/
;/* If you want to use the program or the program referenced in the  */
;/* article, please specify in which data and procedures from STC    */
;/*---------------------------------------------------------------------------*/

;/*    本程序经过测试完全正常, 不提供电话技术支持, 如不能理解, 请自行补充相关基础.  */

;/**************  本程序功能说明  **************

;用STC的MCU的SPI方式控制74HC595驱动8位数码管。

;用户可以修改宏来选择时钟频率, 可以修改寄存器定义是STC12C5A60S2系列 还是 STC12C5628AD
STC12C5410AD  STC12C4052AD系列.

;用户可以在显示函数里修改成共阴或共阳.推荐尽量使用共阴数码管.

;显示效果为: 8个数码管循环显示0,1,2...,A,B..F,消隐.

;*******************************************/

;定义Timer0 1ms重装值
D_Timer0_Reload         EQU     (0-921)         ;1ms for 11.0592MHZ
;D_Timer0_Reload        EQU     (0-1832)        ;1ms for 22.1184MHZ

SPSTAT          DATA    0CDH                    ;STC12C5A60S2系列
SPCTL           DATA    0CEH                    ;STC12C5A60S2系列
SPDAT           DATA    0CFH                    ;STC12C5A60S2系列

;SPSTAT         DATA    084H            ;STC12C5628AD  STC12C5410AD  STC12C4052AD系列
;SPCTL          DATA    085H            ;STC12C5628AD  STC12C5410AD  STC12C4052AD系列
;SPDAT          DATA    086H            ;STC12C5628AD  STC12C5410AD  STC12C4052AD系列

;/**************  本地变量声明  **************/
SPI_SCL         BIT     P1^5    ;SPI同步时钟    P_HC595_SRCLK    pin11    SRCLK    Shift data clock
;SPI_MISO       BIT     P1^6    ;SPI同步数据输入            本例不用
SPI_MOSI        BIT     P1^3    ;SPI同步数据输出    P_HC595_SER    pin 14    SER    data input
P_HC595_RCLK    BIT     P4^1    ;SPI片选(任意IO)  pin 12    RCLk    store (latch) clock

LED8            EQU     030H
display_index   DATA    038H
FLAG0           DATA    20H
B_1ms           BIT     FLAG0.0
```

STC MCU Limited

```
;SPCTL          SPI控制寄存器
;  7     6     5     4     3     2     1     0     Reset Value
;        SSIG  SPEN  DORD  MSTR  CPOL  CPHA  SPR1  SPR0        0x00
SSIG    EQU     1                       ;1: 忽略SS脚，由MSTR位决定主机还是从机
                                        ;0: SS脚用于决定主从机。
SPEN    EQU     1                       ;1: 允许SPI，
                                        ;0：禁止SPI，所有SPI管脚均为普通IO
DORD    EQU     0                       ;1：LSB先发，
                                        ;0：MSB先发
MSTR    EQU     1                       ;1：设为主机
                                        ;0：设为从机
CPOL    EQU     1                       ;1: 空闲时SCLK为高电平，
                                        ;0：空闲时SCLK为低电平
CPHA    EQU     1                       ;
SPR1    EQU     0                       ;SPR1,SPR0   00: fosc/4,    01: fosc/16
SPR0    EQU     0                       ;          10: fosc/64,   11: fosc/128
SPEED_4         EQU     0               ; fosc/4
SPEED_16        EQU     1               ; fosc/16
SPEED_64        EQU     2               ; fosc/64
SPEED_128       EQU     3               ; fosc/128

;SPSTAT         SPI状态寄存器
;  7     6     5  4  3  2  1  0        Reset Value
;        SPIF  WCOL  -     -     -     -     -     -
SPIF    EQU     80H                     ;SPI传输完成标志。写入1清0。
WCOL    EQU     40H                     ;SPI写冲突标志。写入1清0。

;*******************************************************************
;*******************************************************************
        ORG     00H                             ;reset
        LJMP    F_MAIN_FUNC

        ORG     03H                             ;INT0 interrupt
;       LJMP    F_INT0_interrupt
        RETI

        ORG     0BH                             ;Timer0 interrupt
        LJMP    F_Timer0_interrupt
        RETI

        ORG     13H                             ;INT1 interrupt
;       LJMP    F_INT1_interrupt

        ORG     1BH                             ;Timer1 interrupt
;       LJMP    F_Timer1_interrupt
        RETI

;*******************************************************************
;*******************************************************************
```

STC MCU Limited

```
;/*********************************************/
F_MAIN_FUNC:
        MOV     SP,     #50H
        MOV     SPCTL,  #((SSIG SHL 7) + (SPEN SHL 6) + (DORD SHL 5) + (MSTR SHL 4) +
                          (CPOL SHL 3) + (CPHA SHL 2) + SPEED_4);                    //配置SPI

        MOV     TMOD,   #01H                            ;Timer 0 config as 16bit timer, 12T
        MOV     TH0,    #HIGH   D_Timer0_Reload         ;1ms
        MOV     TL0,    #LOW    D_Timer0_Reload
        SETB    ET0     ;
        SETB    TR0     ;
        SETB    EA      ;

        MOV     R0,     #LED8
L_InitLoop1:
        MOV     @R0,    #10H                            ;上电消隐
        INC     R0
        MOV     A,      R0
        CJNE    A,      #(LED8+8),      L_InitLoop1

        MOV     R2,     #HIGH 500                       ;500ms
        MOV     R3,     #LOW  500
        MOV     R4,     #0
L_MainLoop:
        JNB     B_1ms,  $                               ;等待1ms到
        CLR     B_1ms                                   ;清1ms标志

        MOV     A,      R3
        CLR     C
        SUBB    A,      #1
        MOV     R3,     A
        MOV     A,      R2
        SUBB    A,      #0
        MOV     R2,     A
        ORL     A,      R3
        JNZ     L_MainLoop

        MOV     R2,     #HIGH 500                       ;500ms
        MOV     R3,     #LOW  500

        MOV     R0,     #LED8                           ;刷新显示
L_OptionLoop1:
        MOV     A,      R4
        MOV     @R0,    A       ;
        INC     R0
        MOV     A,      R0
        CJNE    A,      #(LED8+8),      L_OptionLoop1
        INC     R4      ;
```

STC MCU Limited

```
        MOV     A,      R4
        CJNE    A,      #11H,L_MainLoop                 ;8个数码管循环显示0,1,2...,A,B..F,消隐.
        MOV     R4,     #0
        SJMP    L_MainLoop

;/*********************************************/
t_display:
;0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  消隐
DB 03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H,000H
;段码

T_COM:
        DB      01H,02H,04H,08H,10H,20H,40H,80H         ;位码

;/*********************************************/
F_SPI_SendByte:                                         ;SPI发送一个字节
        MOV     SPSTAT, #(SPIF + WCOL)                  ;清0 SPIF和WCOL标志
        MOV     SPDAT,  A                               ;发送一个字节
L_SPI_SendByteWait:                                     ;等待发送完成
        MOV     A,      SPSTAT
        ANL     A,      #SPIF
        JZ      L_SPI_SendByteWait
        MOV     SPSTAT, #(SPIF + WCOL)                  ;清0 SPIF和WCOL标志
        RET

;/*********************************************/
F_DisplayScan:                                          ;显示扫描函数
        MOV     DPTR,   #T_COM
        MOV     A,      display_index
        MOVC    A,      @A+DPTR
;       CPL             A                               ;共阴      共阳时注释掉本句
        LCALL   F_SPI_SendByte                          ;输出位码

        MOV     DPTR,   #t_display
        MOV     A,      #LED8
        ADD     A,      display_index
        MOV     R0,     A
        MOV     A,      @R0
        MOVC    A,      @A+DPTR
        CPL     A                                       ;共阳      共阴时注释掉本句
        LCALL   F_SPI_SendByte                          ;输出段码
        SETB    P_HC595_RCLK
        CLR     P_HC595_RCLK                            ;锁存输出数据
        INC     display_index
        MOV     A,      display_index
        CJNE    A,      #8,L_QuitDisplayScan
        MOV     display_index,      #0                  ;8位结束回0
L_QuitDisplayScan:
        RET
```

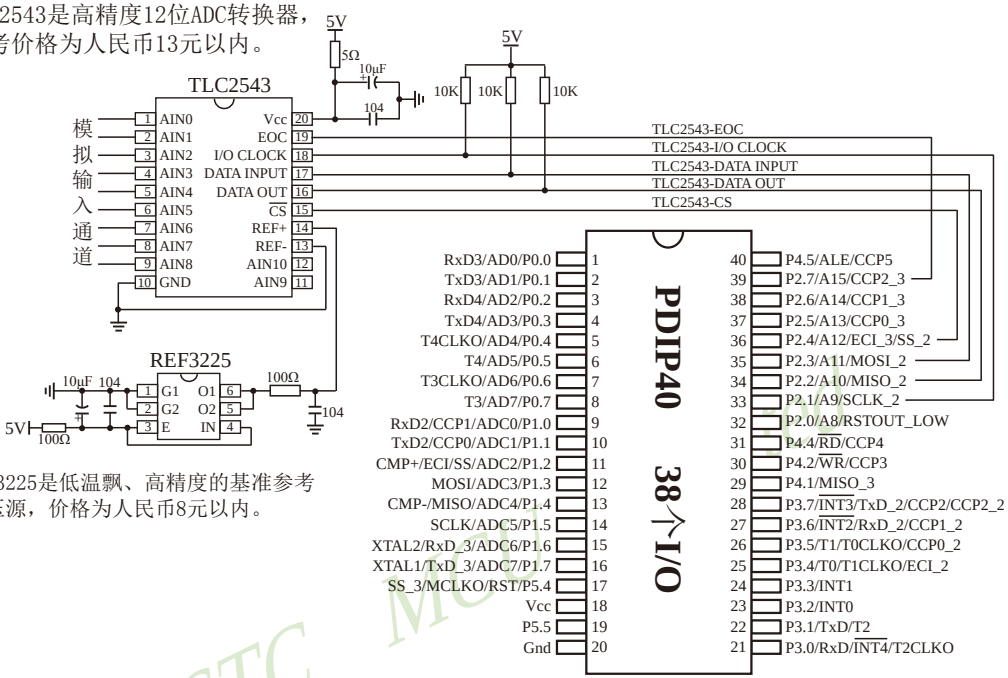
STC MCU Limited

```
;*************************************************************************
;*************************************************************************
F_Timer0_interrupt:                                     ;Timer0 1ms中断函数
        PUSH    PSW                                     ;现场保护
        PUSH    ACC
        MOV     A,      R0
        PUSH    ACC
        PUSH    DPH
        PUSH    DPL

        MOV     TH0,    #HIGH D_Timer0_Reload           ;1ms    重装定时值
        MOV     TL0,    #LOW  D_Timer0_Reload

        LCALL   F_DisplayScan                           ;1ms扫描显示一位
        SETB    B_1ms                                   ;1ms标志

L_QuitT0Interrupt:
        POP     DPL                                     ;现场恢复
        POP     DPH
        POP     ACC
        MOV     R0,     A
        POP     ACC
        POP     PSW
        RETI


        END
```

STC MCU Limited

14.7 利用SPI接口扩展12位ADC(TLC2543)的应用线路图

TLC2543是高精度12位ADC转换器，参考价格为人民币13元以内。



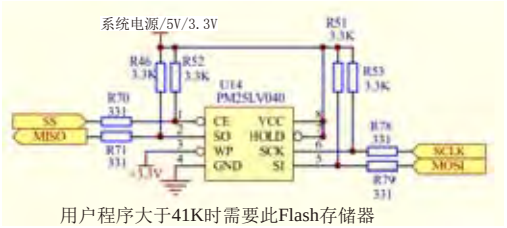
REF3225是低温飘、高精度的基准参考电压源，价格为人民币8元以内。

STC MCU

14.8 利用STC15系列单片机SPI的主模式读写外部串行Flash

14.8.1 利用STC15系列SPI的主模式读写外部串行Flash的参考电路图

Flash控制部分参考电路图



用户程序大于41K时需要此Flash存储器

14.8.2 利用STC15系列SPI的主模式读写外部串行Flash的测试程序

14.8.2.1 通过中断方式利用SPI的主模式读写外部串行Flash的测试程序(C和汇编)

1、C语言程序

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 SPI的主模式读写外部串行Flash举例(中断方式)-------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8058芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz
//本示例所读写的目标Flash为PM25LV040,本代码已使用U7编程器测试通过

#include "reg51.h"

typedef bit BOOL;
typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned long DWORD;

#define FOSC            18432000L
#define BAUD            (65536 - FOSC / 4 / 115200)
```

STC MCU Limited

```
#define  NULL    0
#define  FALSE   0
#define  TRUE    1

sfr      AUXR    =      0x8e;          //辅助寄存器
sfr      P_SW1   =      0xa2;          //外设功能切换寄存器1
#define  SPI_S0         0x04
#define  SPI_S1         0x08

sfr      SPSTAT  =      0xcd;          //SPI状态寄存器
#define  SPIF           0x80           //SPSTAT.7
#define  WCOL           0x40           //SPSTAT.6
sfr      SPCTL   =      0xce;          //SPI控制寄存器
#define  SSIG           0x80           //SPCTL.7
#define  SPEN           0x40           //SPCTL.6
#define  DORD           0x20           //SPCTL.5
#define  MSTR           0x10           //SPCTL.4
#define  CPOL           0x08           //SPCTL.3
#define  CPHA           0x04           //SPCTL.2
#define  SPDHH          0x00           //CPU_CLK/4
#define  SPDH           0x01           //CPU_CLK/16
#define  SPDL           0x02           //CPU_CLK/64
#define  SPDLL          0x03           //CPU_CLK/128
sfr      SPDAT   =      0xcf;          //SPI数据寄存器

sbit     SS      =      P2^4;          //SPI的SS脚,连接到Flash的CE

sfr      IE2     =      0xAF;          //中断控制寄存器2
#define  ESPI           0x02           //IE2.1

#define  SFC_WREN       0x06          //串行Flash命令集
#define  SFC_WRDI       0x04
#define  SFC_RDSR       0x05
#define  SFC_WRSR       0x01
#define  SFC_READ       0x03
#define  SFC_FASTREAD   0x0B
#define  SFC_RDID       0xAB
#define  SFC_PAGEPROG   0x02
#define  SFC_RDCR       0xA1
#define  SFC_WRCR       0xF1
#define  SFC_SECTORER   0xD7
#define  SFC_BLOCKER    0xD8
#define  SFC_CHIPER     0xC7
```

STC MCU Limited

```
void InitUart();
void SendUart(BYTE dat);
void InitSpi();
BYTE SpiShift(BYTE dat);
BOOL FlashCheckID();
BOOL IsFlashBusy();
void FlashWriteEnable();
void FlashErase();
void FlashRead(DWORD addr, DWORD size, BYTE *buffer);
void FlashWrite(DWORD addr, DWORD size, BYTE *buffer);

#define BUFFER_SIZE     1024                    //缓冲区大小
#define TEST_ADDR       0                       //Flash测试地址

BYTE xdata g_Buffer[BUFFER_SIZE];               //Flash读写缓冲区
BOOL g_fFlashOK;                                //Flash状态
BOOL g_fSpiBusy;                                //SPI的工作状态

void main()
{
        int i;

        //初始化Flash状态
        g_fFlashOK = FALSE;
        g_fSpiBusy = FALSE;

        //初始化串口和SPI
        InitUart();
        InitSpi();

        //使能SPI传输中断
        IE2 |= ESPI;
        EA = 1;

        //检测Flash状态
        FlashCheckID();

        //擦除Flash
        FlashErase();
        //读取测试地址的数据
        FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);
        //发送到串口
        for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);
```

STC MCU Limited

```
        //修改置缓冲区
        for (i=0; i<BUFFER_SIZE; i++) g_Buffer[i] = i>>2;
        //将缓冲区的数据写到Flash中
        FlashWrite(TEST_ADDR, BUFFER_SIZE, g_Buffer);

        //读取测试地址的数据
        FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);
        //发送到串口
        for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);

        FlashErase();
        //读取测试地址的数据
        FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);
        //发送到串口
        for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);

        //修改置缓冲区
        for (i=0; i<BUFFER_SIZE; i++) g_Buffer[i]= 255-(i>>2);
        //将缓冲区的数据写到Flash中
        FlashWrite(TEST_ADDR, BUFFER_SIZE, g_Buffer);

        //读取测试地址的数据
        FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);
        //发送到串口
        for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);

        while (1);
}

/*************************************************
SPI中断服务程序
*************************************************/
void spi_isr() interrupt 9 using 1
{
        SPSTAT = SPIF | WCOL;                   //清除SPI状态位
        g_fSpiBusy = FALSE;
}

/*************************************************
串口初始化
入口参数: 无
出口参数: 无
*************************************************/
void InitUart()
```

STC MCU Limited

```
{
	AUXR = 0x40;			//设置定时器1为1T模式
	TMOD = 0x00;			//定时器1为16位重载模式
	TH1 = BAUD >> 8;		//设置波特率
	TL1 = BAUD;
	TR1 = 1;
	SCON = 0x5a;			//设置串口为8位数据位,波特率可变模式
}

/*************************************************
发送数据到串口
入口参数:
  dat : 准备发送的数据
出口参数: 无
*************************************************/
void SendUart(BYTE dat)
{
	while (!TI);			//等待上一个数据发送完成
	TI = 0;				//清除发送完成标志
	SBUF = dat;			//触发本次的数据发送
}

/*************************************************
SPI初始化
入口参数: 无
出口参数: 无
*************************************************/
void InitSpi()
{
//	ACC = P_SW1;			//切换到第一组SPI
//	ACC &= ~(SPI_S0 | SPI_S1);	//SPI_S0=0 SPI_S1=0
//	P_SW1 = ACC;			//(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)

	ACC = P_SW1;			//可用于测试U7,U7使用的是第二组SPI控制Flash
	ACC &= ~(SPI_S0 | SPI_S1);	//SPI_S0=1 SPI_S1=0
	ACC |= SPI_S0;			//(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
	P_SW1 = ACC;

//	ACC = P_SW1;			//切换到第三组SPI
//	ACC &= ~(SPI_S0 | SPI_S1);	//SPI_S0=0 SPI_S1=1
//	ACC |= SPI_S1;			//(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
//	P_SW1 = ACC;

	SPSTAT = SPIF | WCOL;		//清除SPI状态
```

STC MCU Limited

```
	SS = 1;
	SPCTL = SSIG | SPEN | MSTR;				//设置SPI为主模式
}

/*************************************************
使用SPI方式与Flash进行数据交换
入口参数:
  dat : 准备写入的数据
出口参数:
  从Flash中读出的数据
*************************************************/
BYTE SpiShift(BYTE dat)
{
	g_fSpiBusy = TRUE;
	SPDAT = dat;					//触发SPI发送
	while (g_fSpiBusy);				//等待SPI数据传输完成

	return SPDAT;
}

/*************************************************
检测Flash是否准备就绪
入口参数: 无
出口参数:
  0 : 没有检测到正确的Flash
  1 : Flash准备就绪
*************************************************/
BOOL FlashCheckID()
{
	BYTE dat1, dat2;

	SS = 0;
	SpiShift(SFC_RDID);				//发送读取ID命令
	SpiShift(0x00);					//空读3个字节
	SpiShift(0x00);
	SpiShift(0x00);
	dat1 = SpiShift(0x00);				//读取制造商ID1
	SpiShift(0x00);					//读取设备ID
	dat2 = SpiShift(0x00);				//读取制造商ID2
	SS = 1;
							//检测是否为PM25LVxx系列的Flash
	g_fFlashOK = ((dat1 == 0x9d) && (dat2 == 0x7f));

	return g_fFlashOK;
}
```

STC MCU Limited

```
/*************************************************
检测Flash的忙状态
入口参数: 无
出口参数:
  0 : Flash处于空闲状态
  1 : Flash处于忙状态
*************************************************/
BOOL IsFlashBusy()
{
	BYTE dat;

	SS = 0;
	SpiShift(SFC_RDSR);			//发送读取状态命令
	dat = SpiShift(0);			//读取状态
	SS = 1;

	return (dat & 0x01);			//状态值的Bit0即为忙标志
}

/*************************************************
使能Flash写命令
入口参数: 无
出口参数: 无
*************************************************/
void FlashWriteEnable()
{
	while (IsFlashBusy());			//Flash忙检测
	SS = 0;
	SpiShift(SFC_WREN);			//发送写使能命令
	SS = 1;
}

/*************************************************
擦除整片Flash
入口参数: 无
出口参数: 无
*************************************************/
void FlashErase()
{
	if (g_fFlashOK)
	{
		FlashWriteEnable();		//使能Flash写命令
		SS = 0;
```

STC MCU Limited

```
                SpiShift(SFC_CHIPER);                   //发送片擦除命令
                SS = 1;
        }
}

/*************************************************
从Flash中读取数据
入口参数:
  addr  : 地址参数
  size  : 数据块大小
  buffer : 缓冲从Flash中读取的数据
出口参数:
  无
*************************************************/
void FlashRead(DWORD addr, DWORD size, BYTE *buffer)
{
        if (g_fFlashOK)
        {
                while (IsFlashBusy());                  //Flash忙检测
                SS = 0;
                SpiShift(SFC_FASTREAD);                 //使用快速读取命令
                SpiShift(((BYTE *)&addr)[1]);           //设置起始地址
                SpiShift(((BYTE *)&addr)[2]);
                SpiShift(((BYTE *)&addr)[3]);
                SpiShift(0);                            //需要空读一个字节
                while (size)
                {
                        *buffer = SpiShift(0);          //自动连续读取并保存
                        addr++;
                        buffer++;
                        size--;
                }
                SS = 1;
        }
}

/*************************************************
写数据到Flash中
入口参数:
  addr  : 地址参数
  size  : 数据块大小
  buffer : 缓冲需要写入Flash的数据
出口参数: 无
*************************************************/
```

STC MCU Limited

```
void FlashWrite(DWORD addr, DWORD size, BYTE *buffer)
{
    if (g_fFlashOK)
    while (size)
    {
        FlashWriteEnable();                 //使能Flash写命令
        SS = 0;
        SpiShift(SFC_PAGEPROG);             //发送页编程命令
        SpiShift(((BYTE *)&addr)[1]);       //设置起始地址
        SpiShift(((BYTE *)&addr)[2]);
        SpiShift(((BYTE *)&addr)[3]);
        while (size)
        {
            SpiShift(*buffer);              //连续页内写
            addr++;
            buffer++;
            size--;
            if ((addr & 0xff) == 0) break;
        }
        SS = 1;
    }
}
```

STC MCU Limited

2、汇编程序

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 SPI的主模式读写外部串行Flash举例(中断方式)-------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8058芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz
//本示例所读写的目标Flash为PM25LV040,本代码已使用U7编程器测试通过

//BAUD  EQU   0FFE8H          //(65536 - 11059200 / 4 / 115200)
BAUD    EQU   0FFD8H          //(65536 - 18432000 / 4 / 115200)
//BAUD  EQU   0FFD0H          //(65536 - 22118400 / 4 / 115200)

AUXR    DATA  08EH            //辅助寄存器
P_SW1   DATA  0A2H            //外设功能切换寄存器1
SPI_S0  EQU   04H
SPI_S1  EQU   08H

SPSTAT  DATA  0CDH            //SPI状态寄存器
SPIF    EQU   080H            //SPSTAT.7
WCOL    EQU   040H            //SPSTAT.6
SPCTL   DATA  0CEH            //SPI控制寄存器
SSIG    EQU   080H            //SPCTL.7
SPEN    EQU   040H            //SPCTL.6
DORD    EQU   020H            //SPCTL.5
MSTR    EQU   010H            //SPCTL.4
CPOL    EQU   008H            //SPCTL.3
CPHA    EQU   004H            //SPCTL.2
SPDHH   EQU   000H            //CPU_CLK/4
SPDH    EQU   001H            //CPU_CLK/16
SPDL    EQU   002H            //CPU_CLK/64
SPDLL   EQU   003H            //CPU_CLK/128
SPDAT   DATA  0CFH            //SPI数据寄存器

SS      BIT   P2.4            //SPI的SS脚,连接到Flash的CE

IE2     DATA  0AFH            //中断控制寄存器2
```

STC MCU Limited

```
ESPI            EQU     002H                    //IE2.1

SFC_WREN        EQU     0x06                    //串行Flash命令集
SFC_WRDI        EQU     0x04
SFC_RDSR        EQU     0x05
SFC_WRSR        EQU     0x01
SFC_READ        EQU     0x03
SFC_FASTREAD    EQU     0x0B
SFC_RDID        EQU     0xAB
SFC_PAGEPROG    EQU     0x02
SFC_RDCR        EQU     0xA1
SFC_WRCR        EQU     0xF1
SFC_SECTORER    EQU     0xD7
SFC_BLOCKER     EQU     0xD8
SFC_CHIPER      EQU     0xC7

BUFFER_SIZE     EQU     1024                    //缓冲区大小
TEST_ADDR       EQU     0                       //Flash测试地址

G_BUFFER        XDATA   0                       //Flash读写缓冲区
G_FLASHOK       BIT     20H.0                   //Flash状态
G_SPIBUSY       BIT     20H.1                   //SPI的工作状态

ADDR            DATA    21H                     //地址变量,3字节
SIZE            DATA    24H                     //大小变量,3字节

        ORG     0000H
        LJMP    MAIN

        ORG     004BH
        LJMP    SPI_ISR

        ORG     0100H
MAIN:
        MOV     SP,     #3FH
        CLR     G_FLASHOK                       //初始化Flash状态
        CLR     G_SPIBUSY

        LCALL   INITUART                        //初始化串口和SPI
        LCALL   INITSPI

        ORL     IE2,    #ESPI                   //使能SPI中断
        SETB    EA
```

STC MCU Limited

```
LCALL   FLASHCHECKID                    //检测Flash状态

LCALL   FLASHERASE                      //擦除Flash
MOV     ADDR+2,         #0
MOV     ADDR+1,         #HIGH TEST_ADDR
MOV     ADDR+0,         #LOW TEST_ADDR
MOV     SIZE+2,         #0
MOV     SIZE+1,         #HIGH BUFFER_SIZE
MOV     SIZE+0,         #LOW BUFFER_SIZE
MOV     DPTR,           #G_BUFFER
LCALL   FLASHREAD                       //读取测试地址的数据
MOV     SIZE+2,         #0
MOV     SIZE+1,         #HIGH BUFFER_SIZE
MOV     SIZE+0,         #LOW BUFFER_SIZE
MOV     DPTR,           #G_BUFFER
LCALL   SENDUARTBLOCK                   //发送到串口

MOV     SIZE+2,         #0
MOV     SIZE+1,         #HIGH BUFFER_SIZE
MOV     SIZE+0,         #LOW BUFFER_SIZE
MOV     DPTR,           #G_BUFFER
MOV     B,#55H
LCALL   XMEMSET                         //修改置缓冲区
MOV     ADDR+2,         #0
MOV     ADDR+1,         #HIGH TEST_ADDR
MOV     ADDR+0,         #LOW TEST_ADDR
MOV     SIZE+2,         #0
MOV     SIZE+1,         #HIGH BUFFER_SIZE
MOV     SIZE+0,         #LOW BUFFER_SIZE
MOV     DPTR,           #G_BUFFER
LCALL   FLASHWRITE                      //将缓冲区的数据写到Flash中

MOV     ADDR+2,         #0
MOV     ADDR+1,         #HIGH TEST_ADDR
MOV     ADDR+0,         #LOW TEST_ADDR
MOV     SIZE+2,         #0
MOV     SIZE+1,         #HIGH BUFFER_SIZE
MOV     SIZE+0,         #LOW BUFFER_SIZE
MOV     DPTR,           #G_BUFFER
LCALL   FLASHREAD                       //读取测试地址的数据
MOV     SIZE+2,         #0
MOV     SIZE+1,         #HIGH BUFFER_SIZE
MOV     SIZE+0,         #LOW BUFFER_SIZE
MOV     DPTR,           #G_BUFFER
```

STC MCU Limited

```
    LCALL   SENDUARTBLOCK                           //发送到串口

    LCALL   FLASHERASE                              //擦除Flash
    MOV     ADDR+2,         #0
    MOV     ADDR+1,         #HIGH TEST_ADDR
    MOV     ADDR+0,         #LOW TEST_ADDR
    MOV     SIZE+2,         #0
    MOV     SIZE+1,         #HIGH BUFFER_SIZE
    MOV     SIZE+0,         #LOW BUFFER_SIZE
    MOV     DPTR,           #G_BUFFER
    LCALL   FLASHREAD                               //读取测试地址的数据
    MOV     SIZE+2,         #0
    MOV     SIZE+1,         #HIGH BUFFER_SIZE
    MOV     SIZE+0,         #LOW BUFFER_SIZE
    MOV     DPTR,           #G_BUFFER
    LCALL   SENDUARTBLOCK                           //发送到串口

    MOV     SIZE+2,         #0
    MOV     SIZE+1,         #HIGH BUFFER_SIZE
    MOV     SIZE+0,         #LOW BUFFER_SIZE
    MOV     DPTR,           #G_BUFFER
    MOV     B,              #0AAH
    LCALL   XMEMSET                                 //修改置缓冲区
    MOV     ADDR+2,         #0
    MOV     ADDR+1,         #HIGH TEST_ADDR
    MOV      ADDR+0,        #LOW TEST_ADDR
    MOV     SIZE+2,         #0
    MOV     SIZE+1,         #HIGH BUFFER_SIZE
    MOV     SIZE+0,         #LOW BUFFER_SIZE
    MOV     DPTR,           #G_BUFFER
    LCALL   FLASHWRITE                              //将缓冲区的数据写到Flash中

    MOV     ADDR+2,         #0
    MOV     ADDR+1,         #HIGH TEST_ADDR
    MOV     ADDR+0,         #LOW TEST_ADDR
    MOV     SIZE+2,         #0
    MOV     SIZE+1,         #HIGH BUFFER_SIZE
    MOV     SIZE+0,         #LOW BUFFER_SIZE
    MOV     DPTR,           #G_BUFFER
    LCALL   FLASHREAD                               //读取测试地址的数据
    MOV     SIZE+2,         #0
    MOV     SIZE+1,         #HIGH BUFFER_SIZE
    MOV     SIZE+0,         #LOW BUFFER_SIZE
    MOV     DPTR,           #G_BUFFER
    LCALL   SENDUARTBLOCK                           //发送到串口
```

STC MCU Limited

```
        SJMP    $

/***********************************************
SPI中断服务程序
***********************************************/
SPI_ISR:
        MOV     SPSTAT,#SPIF | WCOL             //清除SPI状态位
        CLR     G_SPIBUSY
        RETI

/***********************************************
发送数据到串口
入口参数:
  SIZE  : 数据块大小
  DPTR  : 数据缓冲区
  B     : 设置的值
出口参数: 无
***********************************************/
XMEMSET:
        MOV     A,SIZE
        ORL     A,SIZE+1
        ORL     A,SIZE+2
        JZ      XMSEND
        MOV     A,B
        MOVX    @DPTR,A                         //自动连续设置XDATA数据
        INC     DPTR
        CLR     C
        MOV     A,SIZE+0
        SUBB    A,#1
        MOV     SIZE+0,A
        MOV     A,SIZE+1
        SUBB    A,#0
        MOV     SIZE+1,A
        MOV     A,SIZE+2
        SUBB    A,#0
        MOV     SIZE+2,A
        LJMP    XMEMSET
XMSEND:
        RET

/***********************************************
串口初始化
入口参数: 无
出口参数: 无
```

STC MCU Limited

```
************************************************/
INITUART:
        MOV     AUXR,   #40H                    //设置定时器1为1T模式
        MOV     TMOD,   #00H                    //定时器1为16位重载模式
        MOV     TH1,    #HIGH BAUD              //设置波特率
        MOV     TL1,    #LOW BAUD
        SETB    TR1
        MOV     SCON,   #5AH                    //设置串口为8位数据位,波特率可变模式
        RET

/************************************************
发送数据到串口
入口参数:
  ACC : 准备发送的数据
出口参数: 无
************************************************/
SENDUART:
        JNB     TI,$                            //等待上一个数据发送完成
        CLR     TI                              //清除发送完成标志
        MOV     SBUF,A                          //触发本次的数据发送
        RET

/************************************************
发送数据块到串口
入口参数:
  SIZE  : 数据块大小
  DPTR  : 数据缓冲区
出口参数: 无
************************************************/
SENDUARTBLOCK:
        MOV     A,SIZE
        ORL     A,SIZE+1
        ORL     A,SIZE+2
        JZ      SUBEND
        MOVX    A,@DPTR
        LCALL   SENDUART                        //自动连续发送串口数据
        INC     DPTR
        CLR     C
        MOV     A,SIZE+0
        SUBB    A,#1
        MOV     SIZE+0,A
        MOV     A,SIZE+1
        SUBB    A,#0
        MOV     SIZE+1,A
        MOV     A,SIZE+2
```

STC MCU Limited

```
        SUBB    A,#0
        MOV     SIZE+2,A
        LJMP    SENDUARTBLOCK
SUBEND:
        RET

/*********************************************
SPI初始化
入口参数: 无
出口参数: 无
*********************************************/
INITSPI:
//      MOV     A,P_SW1                         //切换到第一组SPI
//      ANL     A,#NOT (SPI_S0 | SPI_S1)        //SPI_S0=0 SPI_S1=0
//      MOV     P_SW1,A                 //(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)

        MOV     A,P_SW1                         //可用于测试U7,U7使用的是第二组SPI控制Flash
        ANL     A,#NOT (SPI_S0 | SPI_S1)        //SPI_S0=1 SPI_S1=0
        ORL     A,#SPI_S0               //(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
        MOV     P_SW1,A

//      MOV     A,P_SW1                 //切换到第三组SPI
//      ANL     A,#NOT (SPI_S0 | SPI_S1)//SPI_S0=0 SPI_S1=1
//      ORL     A,#SPI_S1               //(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
//      MOV     P_SW1,A

        MOV     SPSTAT,#SPIF | WCOL             //清除SPI状态
        SETB    SS
        MOV     SPCTL,#SSIG | SPEN | MSTR       //设置SPI为主模式
        RET

/*********************************************
使用SPI方式与Flash进行数据交换
入口参数:
  ACC : 准备写入的数据
出口参数:
  ACC : 从Flash中读出的数据
*********************************************/
SPISHIFT:
        SETB    G_SPIBUSY
        MOV     SPDAT,A                         //触发SPI发送
        JB      G_SPIBUSY,$                     //等待SPI数据传输完成
        MOV     A,SPDAT
        RET
```

STC MCU Limited

```
/************************************************
检测Flash是否准备就绪
入口参数: 无
出口参数: CY
  0 : 没有检测到正确的Flash
  1 : Flash准备就绪
************************************************/
FLASHCHECKID:
        CLR     SS
        MOV     A,#SFC_RDID             //发送读取ID命令
        LCALL   SPISHIFT

        LCALL   SPISHIFT                //空读3个字节
        LCALL   SPISHIFT
        LCALL   SPISHIFT

        LCALL   SPISHIFT                //读取制造商ID1
        MOV     R7,A
        LCALL   SPISHIFT                //读取设备ID
        LCALL   SPISHIFT                //读取制造商ID2
        MOV     R6,A
        SETB    SS

        CLR     G_FLASHOK
        CJNE    R7,#9DH,$               //检测是否为PM25LVxx系列的Flash
        CJNE    R6,#7FH,$
        SETB    G_FLASHOK
FLASHERROR:
        MOV     C,G_FLASHOK
        RET
/************************************************
检测Flash的忙状态
入口参数: 无
出口参数: CY
  0 : Flash处于空闲状态
  1 : Flash处于忙状态
************************************************/
ISFLASHBUSY:
        CLR     SS
        MOV     A,#SFC_RDSR             //发送读取状态命令
        LCALL   SPISHIFT
        LCALL   SPISHIFT                //读取状态
        SETB    SS
        MOV     C,ACC.0                 //状态值的Bit0即为忙标志
        RET
```

STC MCU Limited

```
/*************************************************
使能Flash写命令
入口参数: 无
出口参数: 无
*************************************************/
FALSHWRITEENABLE:
        LCALL   ISFLASHBUSY
        JC      $-3                             //Flash忙检测
        CLR     SS
        MOV     A,#SFC_WREN                     //发送写使能命令
        LCALL   SPISHIFT
        SETB    SS
        RET

/*************************************************
擦除整片Flash
入口参数: 无
出口参数: 无
*************************************************/
FLASHERASE:
        JNB     G_FLASHOK,FEREXIT
        LCALL   FALSHWRITEENABLE                //使能Flash写命令
        CLR     SS
        MOV     A,#SFC_CHIPER                   //发送片擦除命令
        LCALL   SPISHIFT
        SETB    SS
FEREXIT:
        RET

/*************************************************
从Flash中读取数据
入口参数:
  ADDR  : FALSH地址参数
  SIZE  : 数据块大小
  DPTR  : 缓冲区首地址
出口参数:
  无
*************************************************/
FLASHREAD:
        JNB     G_FLASHOK,FRDEXIT
        LCALL   ISFLASHBUSY
        JC      $-3                             //Flash忙检测
        CLR     SS
```

STC MCU Limited

```
        MOV     A,#SFC_FASTREAD             //使用快速读取命令
        LCALL   SPISHIFT

        MOV     A,ADDR+2                    //设置起始地址
        LCALL   SPISHIFT
        MOV     A,ADDR+1
        LCALL   SPISHIFT
        MOV     A,ADDR+0
        LCALL   SPISHIFT

        LCALL   SPISHIFT                    //需要空读一个字节
FRDNEXTBYTE:
        MOV     A,SIZE
        ORL     A,SIZE+1
        ORL     A,SIZE+2
        JZ      FRDEND

        LCALL   SPISHIFT                    //自动连续读取并保存
        MOVX    @DPTR,A
        INC     DPTR
        MOV     A,ADDR+0
        ADD     A,#1
        MOV     ADDR+0,A
        MOV     A,ADDR+1
        ADDC    A,#0
        MOV     ADDR+1,A
        MOV     A,ADDR+2
        ADDC    A,#0
        MOV     ADDR+2,A
        CLR     C
        MOV     A,SIZE+0
        SUBB    A,#1
        MOV     SIZE+0,A
        MOV     A,SIZE+1
        SUBB    A,#0
        MOV     SIZE+1,A
        MOV     A,SIZE+2
        SUBB    A,#0
        MOV     SIZE+2,A
        LJMP    FRDNEXTBYTE
FRDEND:
        SETB    SS
FRDEXIT:
        RET
```

STC MCU Limited

```
/*************************************************
写数据到Flash中
入口参数:
   ADDR  :地址参数
   SIZE  :数据块大小
   DPTR  :缓冲需要写入Flash的数据
出口参数: 无
*************************************************/
FLASHWRITE:
        JNB     G_FLASHOK,FWREXIT
FWRNEXTPAGE:
        MOV     A,SIZE
        ORL     A,SIZE+1
        ORL     A,SIZE+2
        JZ      FWREXIT

        LCALL   FALSHWRITEENABLE            //使能Flash写命令
        CLR     SS
        MOV     A,#SFC_PAGEPROG             //发送页编程命令
        LCALL   SPISHIFT

        MOV     A,ADDR+2                    //设置起始地址
        LCALL   SPISHIFT
        MOV     A,ADDR+1
        LCALL   SPISHIFT
        MOV     A,ADDR+0
        LCALL   SPISHIFT

FWRNEXTBYTE:
        MOV     A,SIZE
        ORL     A,SIZE+1
        ORL     A,SIZE+2
        JZ      FRDEND

        MOVX    A,@DPTR
        LCALL   SPISHIFT                    //连续页内写
        INC     DPTR
        MOV     A,ADDR+0
        ADD     A,#1
        MOV     ADDR+0,A
        MOV     A,ADDR+1
        ADDC    A,#0
        MOV     ADDR+1,A
        MOV     A,ADDR+2
        ADDC    A,#0
```

STC MCU Limited

```
        MOV     ADDR+2,A
        CLR     C
        MOV     A,SIZE+0
        SUBB    A,#1
        MOV     SIZE+0,A
        MOV     A,SIZE+1
        SUBB    A,#0
        MOV     SIZE+1,A
        MOV     A,SIZE+2
        SUBB    A,#0
        MOV     SIZE+2,A
        MOV     A,ADDR+0
        JZ      FWREND
        LJMP    FWRNEXTBYTE
FWREND:
        SETB    SS
        LJMP    FWRNEXTPAGE
FWREXIT:
        RET

        END
```

STC MCU Limited

14.8.2.2 通过查询方式利用SPI的主模式读写外部串行Flash的测试程序(C和汇编)

1、C语言程序

```
/*---------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 SPI的主模式读写外部串行Flash举例(查询方式)--------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8058芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz
//本示例所读写的目标Flash为PM25LV040,本代码已使用U7编程器测试通过

#include "reg51.h"

typedef bit BOOL;
typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned long DWORD;

#define FOSC        18432000L
#define BAUD        (65536 - FOSC / 4 / 115200)

#define NULL    0
#define FALSE   0
#define TRUE    1

sfr  AUXR       =       0x8e;       //辅助寄存器
sfr P_SW1       =       0xa2;       //外设功能切换寄存器1
#define SPI_S0          0x04
#define SPI_S1          0x08

sfr SPSTAT      =       0xcd;       //SPI状态寄存器
#define SPIF            0x80        //SPSTAT.7
#define WCOL            0x40        //SPSTAT.6
sfr SPCTL       =       0xce;       //SPI控制寄存器
#define SSIG            0x80        //SPCTL.7
#define SPEN            0x40        //SPCTL.6
#define DORD            0x20        //SPCTL.5
```

STC MCU Limited

```
#define MSTR            0x10            //SPCTL.4
#define CPOL            0x08            //SPCTL.3
#define CPHA            0x04            //SPCTL.2
#define SPDHH           0x00            //CPU_CLK/4
#define SPDH            0x01            //CPU_CLK/16
#define SPDL            0x02            //CPU_CLK/64
#define SPDLL           0x03            //CPU_CLK/128
sfr SPDAT       =       0xcf;           //SPI数据寄存器

sbit SS         =   P2^4;                       //SPI的SS脚,连接到Flash的CE

#define SFC_WREN        0x06            //串行Flash命令集
#define SFC_WRDI        0x04
#define SFC_RDSR        0x05
#define SFC_WRSR        0x01
#define SFC_READ        0x03
#define SFC_FASTREAD    0x0B
#define SFC_RDID        0xAB
#define SFC_PAGEPROG    0x02
#define SFC_RDCR        0xA1
#define SFC_WRCR        0xF1
#define SFC_SECTORER    0xD7
#define SFC_BLOCKER     0xD8
#define SFC_CHIPER      0xC7

void InitUart();
void SendUart(BYTE dat);
void InitSpi();
BYTE SpiShift(BYTE dat);
BOOL FlashCheckID();
BOOL IsFlashBusy();
void FlashWriteEnable();
void FlashErase();
void FlashRead(DWORD addr, DWORD size, BYTE *buffer);
void FlashWrite(DWORD addr, DWORD size, BYTE *buffer);

#define BUFFER_SIZE     1024            //缓冲区大小
#define TEST_ADDR       0               //Flash测试地址

BYTE xdata g_Buffer[BUFFER_SIZE];       //Flash读写缓冲区
BOOL g_fFlashOK;                        //Flash状态

void main()
{
```

STC MCU Limited

```
    int i;

    //初始化Flash状态
    g_fFlashOK = FALSE;

    //初始化串口和SPI
    InitUart();
    InitSpi();

    //检测Flash状态
    FlashCheckID();

    //擦除Flash
    FlashErase();
    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);
    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);

    //修改置缓冲区
    for (i=0; i<BUFFER_SIZE; i++) g_Buffer[i] = i>>2;
    //将缓冲区的数据写到Flash中
    FlashWrite(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);
    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);

    FlashErase();
    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);
    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);

    //修改置缓冲区
    for (i=0; i<BUFFER_SIZE; i++) g_Buffer[i]= 255-(i>>2);
    //将缓冲区的数据写到Flash中
    FlashWrite(TEST_ADDR, BUFFER_SIZE, g_Buffer);

    //读取测试地址的数据
    FlashRead(TEST_ADDR, BUFFER_SIZE, g_Buffer);
    //发送到串口
    for (i=0; i<BUFFER_SIZE; i++) SendUart(g_Buffer[i]);
```

STC MCU Limited

```
    while (1);
}

/*************************************************
串口初始化
入口参数: 无
出口参数: 无
*************************************************/
void InitUart()
{
    AUXR = 0x40;                    //设置定时器1为1T模式
    TMOD = 0x00;                    //定时器1为16位重载模式
    TH1 = BAUD >> 8;                //设置波特率
    TL1 = BAUD;
    TR1 = 1;
    SCON = 0x5a;                    //设置串口为8位数据位,波特率可变模式
}

/*************************************************
发送数据到串口
入口参数:
    dat : 准备发送的数据
出口参数: 无
*************************************************/
void SendUart(BYTE dat)
{
    while (!TI);                    //等待上一个数据发送完成
    TI = 0;                         //清除发送完成标志
    SBUF = dat;                     //触发本次的数据发送
}

/*************************************************
SPI初始化
入口参数: 无
出口参数: 无
*************************************************/
void InitSpi()
{
//  ACC = P_SW1;                            //切换到第一组SPI
//  ACC &= ~(SPI_S0 | SPI_S1);              //SPI_S0=0 SPI_S1=0
//  P_SW1 = ACC;                            //(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)

    ACC = P_SW1;                            //可用于测试U7,U7使用的是第二组SPI控制Flash
    ACC &= ~(SPI_S0 | SPI_S1);              //SPI_S0=1 SPI_S1=0
```

STC MCU Limited

```
    ACC |= SPI_S0;                  //(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
    P_SW1 = ACC;

//  ACC = P_SW1;                    //切换到第三组SPI
//  ACC &= ~(SPI_S0 | SPI_S1);      //SPI_S0=0 SPI_S1=1
//  ACC |= SPI_S1;                  //(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
//  P_SW1 = ACC;

    SPSTAT = SPIF | WCOL;           //清除SPI状态
    SS = 1;
    SPCTL = SSIG | SPEN | MSTR;     //设置SPI为主模式
}

/*************************************************
使用SPI方式与Flash进行数据交换
入口参数:
    dat : 准备写入的数据
出口参数:
    从Flash中读出的数据
*************************************************/
BYTE SpiShift(BYTE dat)
{
    SPDAT = dat;                    //触发SPI发送
    while (!(SPSTAT & SPIF));       //等待SPI数据传输完成
    SPSTAT = SPIF | WCOL;           //清除SPI状态

    return SPDAT;
}

/*************************************************
检测Flash是否准备就绪
入口参数: 无
出口参数:
    0 : 没有检测到正确的Flash
    1 : Flash准备就绪
*************************************************/
BOOL FlashCheckID()
{
    BYTE dat1, dat2;

    SS = 0;
    SpiShift(SFC_RDID);             //发送读取ID命令
    SpiShift(0x00);                 //空读3个字节
    SpiShift(0x00);
    SpiShift(0x00);
```

STC MCU Limited

```
    dat1 = SpiShift(0x00);                  //读取制造商ID1
    SpiShift(0x00);                         //读取设备ID
    dat2 = SpiShift(0x00);                  //读取制造商ID2
    SS = 1;
                                            //检测是否为PM25LVxx系列的Flash
    g_fFlashOK = ((dat1 == 0x9d) && (dat2 == 0x7f));

    return g_fFlashOK;
}

/*************************************************
检测Flash的忙状态
入口参数: 无
出口参数:
    0 : Flash处于空闲状态
    1 : Flash处于忙状态
*************************************************/
BOOL IsFlashBusy()
{
    BYTE dat;

    SS = 0;
    SpiShift(SFC_RDSR);                     //发送读取状态命令
    dat = SpiShift(0);                      //读取状态
    SS = 1;

    return (dat & 0x01);                    //状态值的Bit0即为忙标志
}

/*************************************************
使能Flash写命令
入口参数: 无
出口参数: 无
*************************************************/
void FlashWriteEnable()
{
    while (IsFlashBusy());                  //Flash忙检测
    SS = 0;
    SpiShift(SFC_WREN);                     //发送写使能命令
    SS = 1;
}

/*************************************************
擦除整片Flash
入口参数: 无
```

STC MCU Limited

```
出口参数: 无
**************************************************/
void FlashErase()
{
    if (g_fFlashOK)
    {
        FlashWriteEnable();                 //使能Flash写命令
        SS = 0;
        SpiShift(SFC_CHIPER);               //发送片擦除命令
        SS = 1;
    }
}

/**************************************************
从Flash中读取数据
入口参数:
    addr   : 地址参数
    size   : 数据块大小
    buffer : 缓冲从Flash中读取的数据
出口参数:
    无
**************************************************/
void FlashRead(DWORD addr, DWORD size, BYTE *buffer)
{
    if (g_fFlashOK)
    {
        while (IsFlashBusy());              //Flash忙检测
        SS = 0;
        SpiShift(SFC_FASTREAD);             //使用快速读取命令
        SpiShift(((BYTE *)&addr)[1]);       //设置起始地址
        SpiShift(((BYTE *)&addr)[2]);
        SpiShift(((BYTE *)&addr)[3]);
        SpiShift(0);                        //需要空读一个字节
        while (size)
        {
            *buffer = SpiShift(0);          //自动连续读取并保存
            addr++;
            buffer++;
            size--;
        }
        SS = 1;
    }
}

/**************************************************
```

STC MCU Limited

```
写数据到Flash中
入口参数:
   addr  : 地址参数
   size  : 数据块大小
   buffer : 缓冲需要写入Flash的数据
出口参数: 无
**********************************************/
void FlashWrite(DWORD addr, DWORD size, BYTE *buffer)
{
   if (g_fFlashOK)
   while (size)
   {
      FlashWriteEnable();                          //使能Flash写命令
      SS = 0;
      SpiShift(SFC_PAGEPROG);                      //发送页编程命令
      SpiShift(((BYTE *)&addr)[1]);                //设置起始地址
      SpiShift(((BYTE *)&addr)[2]);
      SpiShift(((BYTE *)&addr)[3]);
      while (size)
      {
         SpiShift(*buffer);                        //连续页内写
         addr++;
         buffer++;
         size--;
         if ((addr & 0xff) == 0) break;
      }
      SS = 1;
   }
}
```

STC MCU Limited

2、汇编程序

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ---------------------------------------------------------------------------*/
/* --- STC15W4K60S4 系列 SPI的主模式读写外部串行Flash举例(查询方式)--------*/
/* --- Mobile: (86)13922805190 -------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 ----------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ----------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -----------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/

//本示例在Keil开发环境下请选择Intel的8058芯片型号进行编译
//假定测试芯片的工作频率为18.432MHz
//本示例所读写的目标Flash为PM25LV040,本代码已使用U7编程器测试通过

//BAUD          EQU   0FFE8H          //(65536 - 11059200 / 4 / 115200)
BAUD            EQU   0FFD8H          //(65536 - 18432000 / 4 / 115200)
//BAUD          EQU   0FFD0H          //(65536 - 22118400 / 4 / 115200)

AUXR            DATA  08EH           //辅助寄存器
P_SW1           DATA  0A2H           //外设功能切换寄存器1
SPI_S0          EQU   04H
SPI_S1          EQU   08H

SPSTAT          DATA  0CDH           //SPI状态寄存器
SPIF            EQU   080H           //SPSTAT.7
WCOL            EQU   040H           //SPSTAT.6
SPCTL           DATA  0CEH           //SPI控制寄存器
SSIG            EQU   080H           //SPCTL.7
SPEN            EQU   040H           //SPCTL.6
DORD            EQU   020H           //SPCTL.5
MSTR            EQU   010H           //SPCTL.4
CPOL            EQU   008H           //SPCTL.3
CPHA            EQU   004H           //SPCTL.2
SPDHH           EQU   000H           //CPU_CLK/4
SPDH            EQU   001H           //CPU_CLK/16
SPDL            EQU   002H           //CPU_CLK/64
SPDLL           EQU   003H           //CPU_CLK/128
SPDAT           DATA  0CFH           //SPI数据寄存器

SS              BIT   P2.4           //SPI的SS脚,连接到Flash的CE

SFC_WREN        EQU   0x06           //串行Flash命令集
```

STC MCU Limited

```
SFC_WRDI        EQU   0x04
SFC_RDSR        EQU   0x05
SFC_WRSR        EQU   0x01
SFC_READ        EQU   0x03
SFC_FASTREAD    EQU   0x0B
SFC_RDID        EQU   0xAB
SFC_PAGEPROG    EQU   0x02
SFC_RDCR        EQU   0xA1
SFC_WRCR        EQU   0xF1
SFC_SECTORER    EQU   0xD7
SFC_BLOCKER     EQU   0xD8
SFC_CHIPER      EQU   0xC7

BUFFER_SIZE     EQU   1024              //缓冲区大小
TEST_ADDR       EQU   0                 //Flash测试地址

G_BUFFER        XDATA 0                 //Flash读写缓冲区
G_FLASHOK       BIT   20H.0             //Flash状态

ADDR            DATA  21H               //地址变量,3字节
SIZE            DATA  24H               //大小变量,3字节

        ORG   0000H
        LJMP  MAIN

        ORG   0100H
MAIN:
        MOV   SP,#3FH
        CLR   G_FLASHOK                 //初始化Flash状态

        LCALL INITUART                  //初始化串口和SPI
        LCALL INITSPI

        LCALL FLASHCHECKID              //检测Flash状态

        LCALL FLASHERASE                //擦除Flash
        MOV   ADDR+2,#0
        MOV   ADDR+1,#HIGH TEST_ADDR
        MOV   ADDR+0,#LOW TEST_ADDR
        MOV   SIZE+2,#0
        MOV   SIZE+1,#HIGH BUFFER_SIZE
        MOV   SIZE+0,#LOW BUFFER_SIZE
        MOV   DPTR,#G_BUFFER
        LCALL FLASHREAD                 //读取测试地址的数据
        MOV   SIZE+2,#0
```

STC MCU Limited

```
MOV    SIZE+1,#HIGH BUFFER_SIZE
MOV    SIZE+0,#LOW BUFFER_SIZE
MOV    DPTR,#G_BUFFER
LCALL  SENDUARTBLOCK                    //发送到串口

MOV    SIZE+2,#0
MOV    SIZE+1,#HIGH BUFFER_SIZE
MOV    SIZE+0,#LOW BUFFER_SIZE
MOV    DPTR,#G_BUFFER
MOV    B,#33H
LCALL  XMEMSET                          //修改置缓冲区
MOV    ADDR+2,#0
MOV    ADDR+1,#HIGH TEST_ADDR
MOV    ADDR+0,#LOW TEST_ADDR
MOV    SIZE+2,#0
MOV    SIZE+1,#HIGH BUFFER_SIZE
MOV    SIZE+0,#LOW BUFFER_SIZE
MOV    DPTR,#G_BUFFER
LCALL  FLASHWRITE                       //将缓冲区的数据写到Flash中

MOV    ADDR+2,#0
MOV    ADDR+1,#HIGH TEST_ADDR
MOV    ADDR+0,#LOW TEST_ADDR
MOV    SIZE+2,#0
MOV    SIZE+1,#HIGH BUFFER_SIZE
MOV    SIZE+0,#LOW BUFFER_SIZE
MOV    DPTR,#G_BUFFER
LCALL  FLASHREAD                        //读取测试地址的数据
MOV    SIZE+2,#0
MOV    SIZE+1,#HIGH BUFFER_SIZE
MOV    SIZE+0,#LOW BUFFER_SIZE
MOV    DPTR,#G_BUFFER
LCALL  SENDUARTBLOCK                    //发送到串口

LCALL  FLASHERASE                       //擦除Flash
MOV    ADDR+2,#0
MOV    ADDR+1,#HIGH TEST_ADDR
MOV    ADDR+0,#LOW TEST_ADDR
MOV    SIZE+2,#0
MOV    SIZE+1,#HIGH BUFFER_SIZE
MOV    SIZE+0,#LOW BUFFER_SIZE
MOV    DPTR,#G_BUFFER
LCALL  FLASHREAD                        //读取测试地址的数据
MOV    SIZE+2,#0
```

STC MCU Limited

```
        MOV    SIZE+1,#HIGH BUFFER_SIZE
        MOV    SIZE+0,#LOW BUFFER_SIZE
        MOV    DPTR,#G_BUFFER
        LCALL  SENDUARTBLOCK                    //发送到串口

        MOV    SIZE+2,#0
        MOV    SIZE+1,#HIGH BUFFER_SIZE
        MOV    SIZE+0,#LOW BUFFER_SIZE
        MOV    DPTR,#G_BUFFER
        MOV    B,#099H
        LCALL  XMEMSET                          //修改置缓冲区
        MOV    ADDR+2,#0
        MOV    ADDR+1,#HIGH TEST_ADDR
        MOV    ADDR+0,#LOW TEST_ADDR
        MOV    SIZE+2,#0
        MOV    SIZE+1,#HIGH BUFFER_SIZE
        MOV    SIZE+0,#LOW BUFFER_SIZE
        MOV    DPTR,#G_BUFFER
        LCALL   FLASHWRITE                      //将缓冲区的数据写到Flash中

        MOV    ADDR+2,#0
        MOV    ADDR+1,#HIGH TEST_ADDR
        MOV    ADDR+0,#LOW TEST_ADDR
        MOV    SIZE+2,#0
        MOV    SIZE+1,#HIGH BUFFER_SIZE
        MOV    SIZE+0,#LOW BUFFER_SIZE
        MOV    DPTR,#G_BUFFER
        LCALL  FLASHREAD                        //读取测试地址的数据
        MOV    SIZE+2,#0
        MOV    SIZE+1,#HIGH BUFFER_SIZE
        MOV    SIZE+0,#LOW BUFFER_SIZE
        MOV    DPTR,#G_BUFFER
        LCALL  SENDUARTBLOCK                    //发送到串口

        SJMP   $

/*************************************************
发送数据到串口
入口参数:
  SIZE  :数据块大小
  DPTR  :数据缓冲区
  B     :设置的值
出口参数: 无
*************************************************/
```

STC MCU Limited

```
XMEMSET:
        MOV     A,SIZE
        ORL     A,SIZE+1
        ORL     A,SIZE+2
        JZ      XMSEND
        MOV     A,B
        MOVX    @DPTR,A                 //自动连续设置XDATA数据
        INC     DPTR
        CLR     C
        MOV     A,SIZE+0
        SUBB    A,#1
        MOV     SIZE+0,A
        MOV     A,SIZE+1
        SUBB    A,#0
        MOV     SIZE+1,A
        MOV     A,SIZE+2
        SUBB    A,#0
        MOV     SIZE+2,A
        LJMP    XMEMSET
XMSEND:
        RET

/*************************************************
串口初始化
入口参数: 无
出口参数: 无
*************************************************/
INITUART:
        MOV     AUXR,#40H               //设置定时器1为1T模式
        MOV     TMOD,#00H               //定时器1为16位重载模式
        MOV     TH1,#HIGH BAUD          //设置波特率
        MOV     TL1,#LOW BAUD
        SETB    TR1
        MOV     SCON,#5AH               //设置串口为8位数据位,波特率可变模式
        RET

/*************************************************
发送数据到串口
入口参数:
  ACC : 准备发送的数据
出口参数: 无
*************************************************/
SENDUART:
        JNB     TI,     $               //等待上一个数据发送完成
        CLR     TI                      //清除发送完成标志
```

STC MCU Limited

```
        MOV     SBUF,   A               //触发本次的数据发送
        RET

/*********************************************
发送数据块到串口
入口参数:
  SIZE  :数据块大小
  DPTR  :数据缓冲区
出口参数: 无
*********************************************/
SENDUARTBLOCK:
        MOV     A,SIZE
        ORL     A,SIZE+1
        ORL     A,SIZE+2
        JZ      SUBEND
        MOVX    A,@DPTR
        LCALL   SENDUART                //自动连续发送串口数据
        INC     DPTR
        CLR     C
        MOV     A,SIZE+0
        SUBB    A,#1
        MOV     SIZE+0,A
        MOV     A,SIZE+1
        SUBB    A,#0
        MOV     SIZE+1,A
        MOV     A,SIZE+2
        SUBB    A,#0
        MOV     SIZE+2,A
        LJMP    SENDUARTBLOCK
SUBEND:
        RET

/*********************************************
SPI初始化
入口参数: 无
出口参数: 无
*********************************************/
INITSPI:
//      MOV     A,P_SW1                 //切换到第一组SPI
//      ANL     A,#NOT (SPI_S0 | SPI_S1)        //SPI_S0=0 SPI_S1=0
//      MOV     P_SW1,A                 //(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)

        MOV     A,P_SW1                 //可用于测试U7,U7使用的是第二组SPI控制Flash
        ANL     A,#NOT (SPI_S0 | SPI_S1)        //SPI_S0=1 SPI_S1=0
        ORL     A,#SPI_S0               //(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
```

STC MCU Limited

```
        MOV     P_SW1,A

//      MOV     A,P_SW1                         //切换到第三组SPI
//      ANL     A,#NOT (SPI_S0 | SPI_S1)        //SPI_S0=0 SPI_S1=1
//      ORL     A,#SPI_S1               //(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
//      MOV     P_SW1,A

        MOV     SPSTAT,#SPIF | WCOL             //清除SPI状态
        SETB    SS
        MOV     SPCTL,#SSIG | SPEN | MSTR       //设置SPI为主模式
        RET

/*************************************************
使用SPI方式与Flash进行数据交换
入口参数:
  ACC : 准备写入的数据
出口参数:
  ACC : 从Flash中读出的数据
*************************************************/
SPISHIFT:
        MOV     SPDAT,A                         //触发SPI发送
WAITSPI:
        MOV     A,SPSTAT                        //等待SPI数据传输完成
        ANL     A,#SPIF
        JZ      WAITSPI
        MOV     SPSTAT,#SPIF | WCOL             //清除SPI状态
        MOV     A,SPDAT
        RET

/*************************************************
检测Flash是否准备就绪
入口参数: 无
出口参数: CY
  0 : 没有检测到正确的Flash
  1 : Flash准备就绪
*************************************************/
FLASHCHECKID:
        CLR     SS
        MOV     A,#SFC_RDID                     //发送读取ID命令
        LCALL   SPISHIFT

        LCALL   SPISHIFT                        //空读3个字节
        LCALL   SPISHIFT
        LCALL   SPISHIFT
```

STC MCU Limited

```
        LCALL   SPISHIFT                //读取制造商ID1
        MOV     R7,A
        LCALL   SPISHIFT                //读取设备ID
        LCALL   SPISHIFT                //读取制造商ID2
        MOV     R6,A
        SETB    SS

        CLR     G_FLASHOK
        CJNE    R7,     #9DH,$          //检测是否为PM25LVxx系列的Flash
        CJNE    R6,     #7FH,$
        SETB    G_FLASHOK
FLASHERROR:
        MOV     C,      G_FLASHOK
        RET

/***********************************************
检测Flash的忙状态
入口参数: 无
出口参数: CY
  0 : Flash处于空闲状态
  1 : Flash处于忙状态
***********************************************/
ISFLASHBUSY:
        CLR     SS
        MOV     A,#SFC_RDSR             //发送读取状态命令
        LCALL   SPISHIFT
        LCALL   SPISHIFT                //读取状态
        SETB    SS
        MOV     C,      ACC.0           //状态值的Bit0即为忙标志
        RET
/***********************************************
使能Flash写命令
入口参数: 无
出口参数: 无
***********************************************/
FALSHWRITEENABLE:
        LCALL   ISFLASHBUSY
        JC      $-3                             //Flash忙检测
        CLR     SS
        MOV     A,      #SFC_WREN               //发送写使能命令
        LCALL   SPISHIFT
        SETB    SS
        RET
```

STC MCU Limited

```
/***********************************************
擦除整片Flash
入口参数: 无
出口参数: 无
***********************************************/
FLASHERASE:
        JNB     G_FLASHOK,FEREXIT
        LCALL   FALSHWRITEENABLE            //使能Flash写命令
        CLR     SS
        MOV     A,#SFC_CHIPER               //发送片擦除命令
        LCALL   SPISHIFT
        SETB    SS
FEREXIT:
        RET

/***********************************************
从Flash中读取数据
入口参数:
  ADDR  : FALSH地址参数
  SIZE  : 数据块大小
  DPTR  : 缓冲区首地址
出口参数:
  无
***********************************************/
FLASHREAD:
        JNB     G_FLASHOK,FRDEXIT
        LCALL   ISFLASHBUSY
        JC      $-3                         //Flash忙检测
        CLR     SS

        MOV     A,#SFC_FASTREAD             //使用快速读取命令
        LCALL   SPISHIFT

        MOV     A,ADDR+2                    //设置起始地址
        LCALL   SPISHIFT
        MOV     A,ADDR+1
        LCALL   SPISHIFT
        MOV     A,ADDR+0
        LCALL   SPISHIFT

        LCALL   SPISHIFT                    //需要空读一个字节
FRDNEXTBYTE:
        MOV     A,SIZE
        ORL    A,SIZE+1
```

STC MCU Limited

```
        ORL    A,SIZE+2
        JZ     FRDEND

        LCALL  SPISHIFT                  //自动连续读取并保存
        MOVX   @DPTR,A
        INC    DPTR
        MOV    A,ADDR+0
        ADD    A,#1
        MOV    ADDR+0,A
        MOV    A,ADDR+1
        ADDC   A,#0
        MOV    ADDR+1,A
        MOV    A,ADDR+2
        ADDC   A,#0
        MOV    ADDR+2,A
        CLR    C
        MOV    A,SIZE+0
        SUBB   A,#1
        MOV    SIZE+0,A
        MOV    A,SIZE+1
        SUBB   A,#0
        MOV    SIZE+1,A
        MOV    A,SIZE+2
        SUBB   A,#0
        MOV    SIZE+2,A
        LJMP   FRDNEXTBYTE
FRDEND:
        SETB   SS
FRDEXIT:
        RET

/************************************************
写数据到Flash中
入口参数:
  ADDR  : 地址参数
  SIZE  : 数据块大小
  DPTR  : 缓冲需要写入Flash的数据
出口参数: 无
************************************************/
FLASHWRITE:
        JNB    G_FLASHOK,FWREXIT
FWRNEXTPAGE:
        MOV    A,SIZE
        ORL    A,SIZE+1
```

STC MCU Limited

```
        ORL     A,SIZE+2
        JZ      FWREXIT

        LCALL   FALSHWRITEENABLE        //使能Flash写命令
        CLR     SS
        MOV     A,#SFC_PAGEPROG         //发送页编程命令
        LCALL   SPISHIFT

        MOV     A,ADDR+2                //设置起始地址
        LCALL   SPISHIFT
        MOV     A,ADDR+1
        LCALL   SPISHIFT
        MOV     A,ADDR+0
        LCALL   SPISHIFT

FWRNEXTBYTE:
        MOV     A,SIZE
        ORL     A,SIZE+1
        ORL     A,SIZE+2
        JZ      FRDEND

        MOVX    A,@DPTR
        LCALL   SPISHIFT                //连续页内写
        INC     DPTR
        MOV     A,ADDR+0
        ADD     A,#1
        MOV     ADDR+0,A
        MOV     A,ADDR+1
        ADDC    A,#0
        MOV     ADDR+1,A
        MOV     A,ADDR+2
        ADDC    A,#0
        MOV     ADDR+2,A
        CLR     C
        MOV     A,SIZE+0
        SUBB    A,#1
        MOV     SIZE+0,A
        MOV     A,SIZE+1
        SUBB    A,#0
        MOV     SIZE+1,A
        MOV     A,SIZE+2
        SUBB    A,#0
        MOV     SIZE+2,A
        MOV     A,ADDR+0
        JZ      FWREND
```

STC MCU Limited

```
        LJMP    FWRNEXTBYTE
FWREND:
        SETB    SS
        LJMP    FWRNEXTPAGE
FWREXIT:
        RET

        END
```

STC MCU Limited

14.9 SPI的特别注意事项(仅针对以15F和15L开头的单片机)

——只支持SPI主机模式，不支持SPI从机模式

STC单片机中以15F和15L开头且有SPI功能的单片机(如STC15F2K60S2型号及STC15L408AD单片机)的SPI从机模式暂不能使用，但它们的SPI主机模式可正常使用。因此，建议用户不要使用以15F和15L开头且有SPI功能的单片机的SPI从机模式。

注意，以15W开头的单片机不存在上述问题，以15W开头且有SPI功能的单片机既支持SPI主机模式，也支持SPI从机模式。如，STC15W408S、STC15W1K16S等型号单片机既支持SPI主机模式，也支持SPI从机模式

第15章 编译器(汇编器)/ISP编程器(烧录)/仿真器说明

15.1 编译器/汇编器的说明及头文件

STC单片机应使用何种编译器/汇编器：

1. 任何老的编译器/汇编器都可以支持，流行用Keil C51

2. 把STC单片机当成Intel的8052/87C52/87C54/87C58或Philips的P87C52/P87C54/P87C58编译，头文件包含<reg51.h>即可。新增特殊功能寄存器用sfr声明，新增特殊功能寄存器位用sbit声明。例如，对要用到的新增P4口特殊功能寄存器及特殊功能寄存器位的地址声明如下：

C语言地址声明：

```
sfr  P4    =  0xC0;     //8 bit Port4     P4.7 P4.6 P4.5 P4.4 P4.3 P4.2 P4.1 P4.0     1111,1111
sfr  P4M0  =  0xB4;     //                                                            0000,0000
sfr  P4M1  =  0xB3;     //                                                            0000,0000

sbit     P40    =    P4^0;
sbit     P41    =    P4^1;
sbit     P42    =    P4^2;
sbit     P43    =    P4^3;
sbit     P44    =    P4^4;
sbit     P45    =    P4^5;
sbit     P46    =    P4^6;
sbit     P47    =    P4^7;
```

汇编语言地址声明：

```
        P4      EQU     0C0H            ; or  P4            DATA    0C0H
        P4M1    EQU     0B3H            ; or  P4M1          DATA    0B3H
        P4M0    EQU     0B4H            ; or  P4M1          DATA    0B4H

        P40     EQU     0C0H
        P41     EQU     0C1H
        P42     EQU     0C2H
        P43     EQU     0C3H
        P44     EQU     0C4H
        P45     EQU     0C5H
        P46     EQU     0C6H
        P47     EQU     0C7H
```

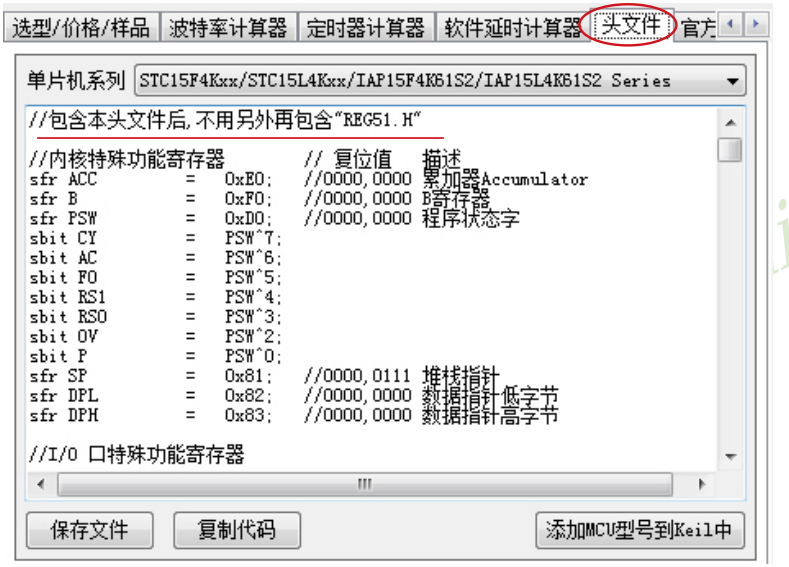
;以上为P4口新增功能寄存器的地址声明

当然如果新增功能寄存器在用户程序中用不到的话，也可以不声明。

STC MCU Limited

注意：如果用户所需包含的头文件不在Keil C的系统目录(C:\keil\C51\INC)下，用""将该头文件名包含进来，如果所需的头文件在Keil C的系统目录下，既可用""，也可用< >包含进来.

对于STC部分单片机，可以到STC官方网站www.STCMCU.com下载用户所使用的相应系列单片机的头文件(如果找不到所需的文件用ctrl+F查找)，STC15系列单片机还可以用最新的ISP下载工具STC-ISP-15xx-V6.58生成相应的头文件并保存，如下图所示。在编译具体STC系列单片机程序时，这些相应的头文件可以代替"reg51.h"。

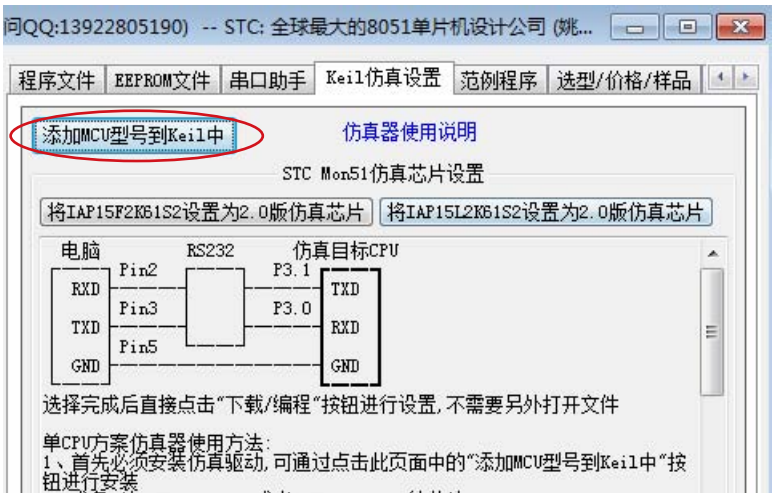


Keil C51集成开发环境有许多版本，而对于8051单片机最常用的版本有Keil μVision2、Keil μVision3及Keil μVision4.

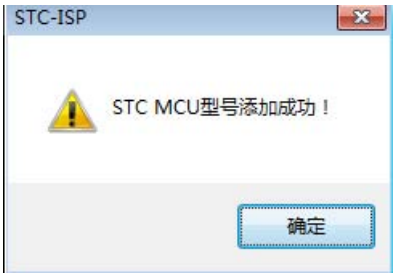
注意：由于STC系列单片机是新发展的芯片，一般情况下在Keil μVision设备库中没有STC系列单片机。在编辑、编译STC系列单片机应用程序时，可选任何厂家的51或52系列单片机，再用汇编或C语言对STC系列单片机新增特殊功能寄存器进行定义，也可以通过STC-ISP下载编程工具将STC型号MCU添加到Keil μVision4或Keil μVision3或Keil μVision2的设备库中。

如果用户需在Keil μVision4或Keil μVision3或Keil μVision2的设备库中增加STC型号MCU，则可按如下步骤进行设置：

① 打开STC-ISP下载编程工具的最新软件STC-ISP-V6.65，选择"Keil仿真设置"页面，点击该页面中的【添加MCU型号到Keil中】按钮。



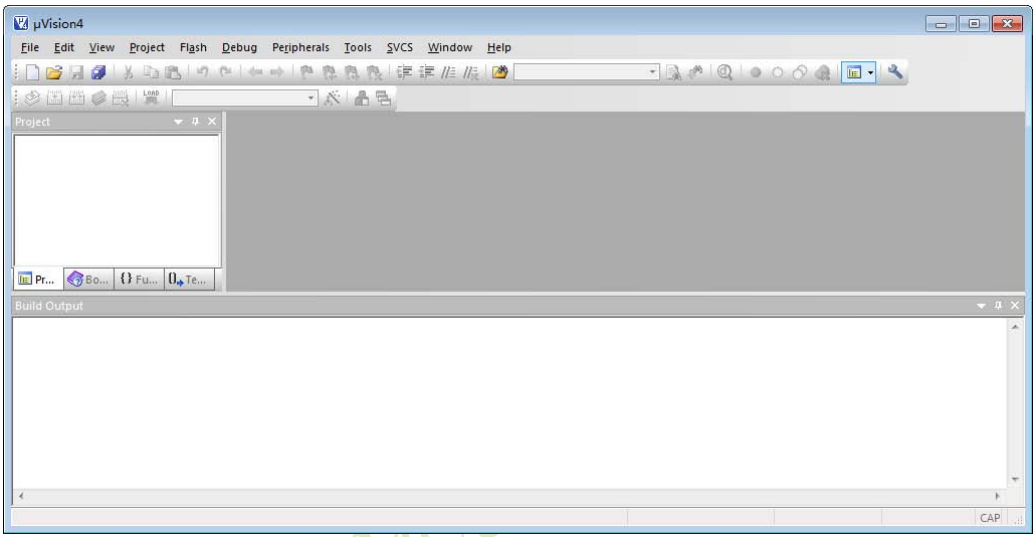
② 在弹出的“浏览文件夹”对话框中选择Keil安装目录（一般可能为“C:\keil”），然后单击【确定】，这样就将STC型号的MCU成功添加到Keil μVision2设备库中了。



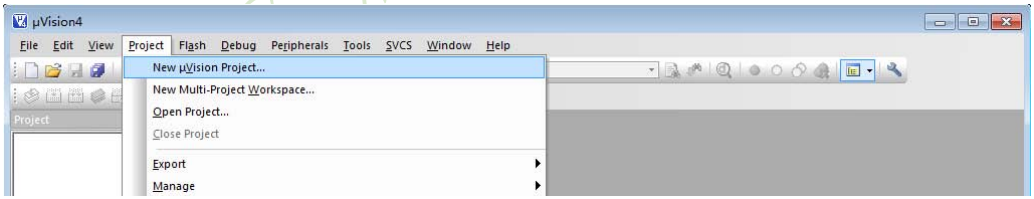
下面以Keil μVision4为例，详细介绍如何使用Keil μVision4开发、编译、调试用户程序。

一、如何新建项目及在所新建的项目中添加STC型号MCU进行开发、编译、调试用户程序：

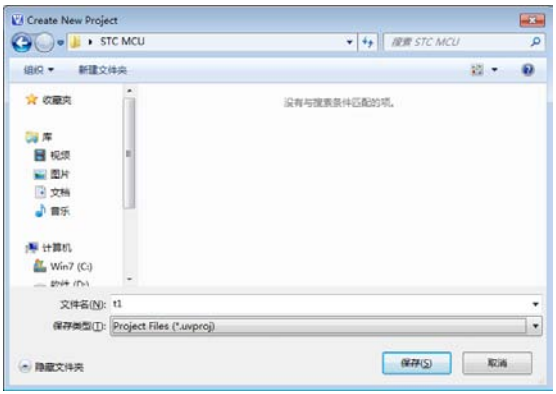
（1）启动Keil μVision4，进入Keil μVision4后的编辑界面如下所示：



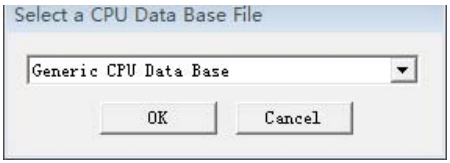
（2）建立一个新工程：单击Project菜单，在弹出的下拉菜单中选中New Project选项

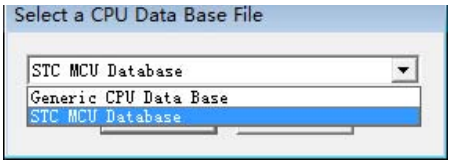


（3）在弹出的对话框中选择新项目要保存的路径和文件名，例如：保存路径为C:\Users\THINK\Documents\STC MCU，项目名为t1，单击保存即可。Keil μVision4的项目文件扩展名为.uvproj

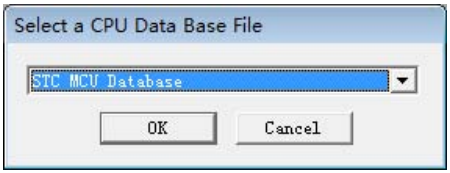


（4）因之前已经通过STC-ISP下载编程工具将STC型号MCU添加到Keil μVision2的设备库中，所以在上一步【保存】之后会弹出“选择设备数据库”的对话框，如下图所示。该“选择设备数据库”的对话框中有“通用CPU数据库(Generic CPU Database)”和“STC MCU数据库(STC MCU Database)”两个选项。

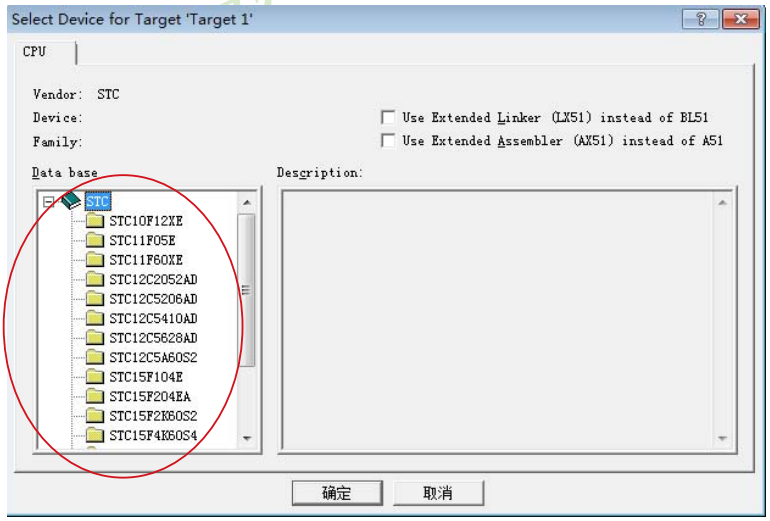




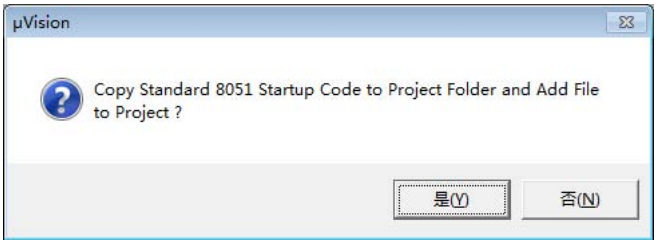
如用户所使用的单片机是STC系列单片机，则在这里选择“STC MCU数据库(STC MCU Database)”，点击【OK】按钮确定。



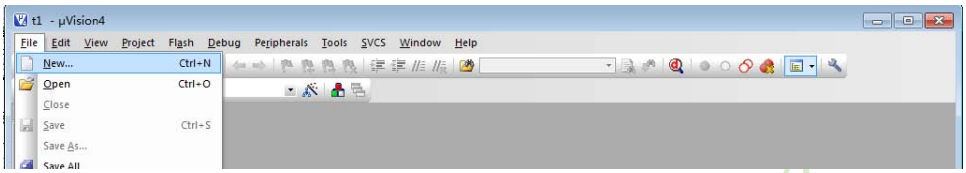
（5）在上一步“选择设备数据库”后会弹出"Select Device for Target"对话框，如下所示。因上一步中我们选择了“STC MCU数据库(STC MCU Database)”，所以这里的MCU型号都是STC型号，用户可在左侧的数据列表(Data base)选择自己所使用的具体单片机型号。



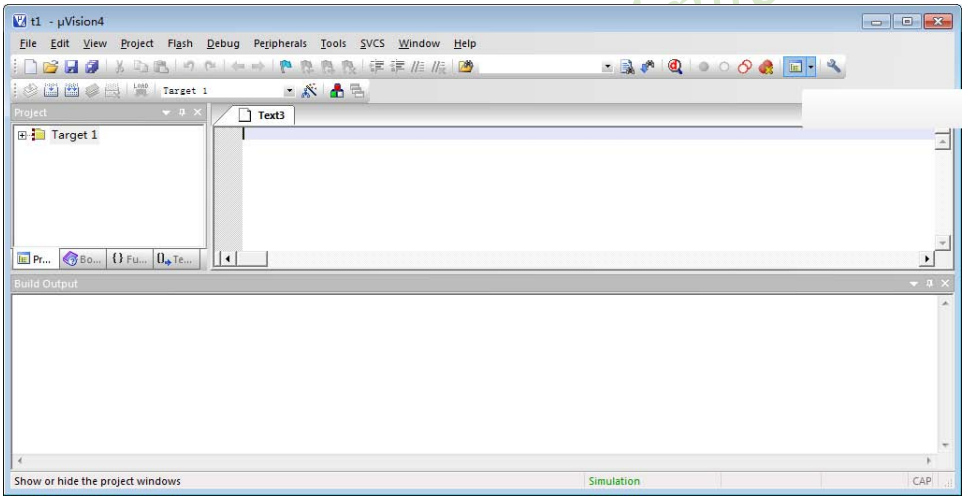
（6）选择好单片机型号并点击确定后，程序会询问是否将标准51初始化程序(STARTUP.51)加入到项目中，如下图所示。选择【是】按钮，程序会自动复制标准51初始化程序到项目所在目录并将其加入项目中。一般情况下，选择【否】按钮



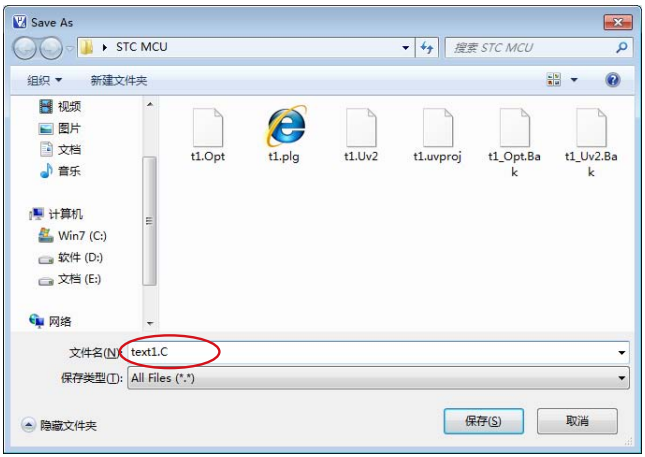
（7）项目建好后开始编写程序了，选择“File”菜单，再在下拉菜单中单击“New”选项



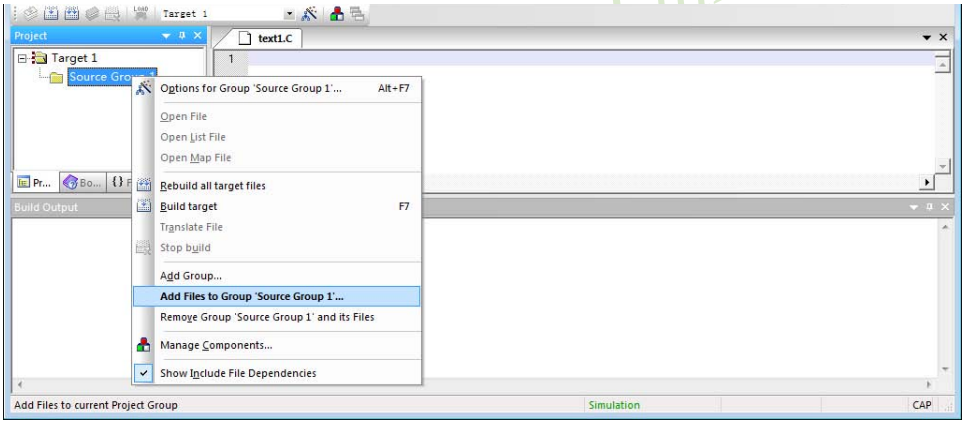
新建文件后界面如下图所示



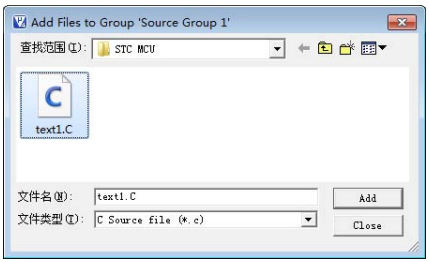
此时光标在编辑窗口里闪烁，这时可以键入用户的应用程序了，输入程序后单击菜单上的“File”，在下拉菜单中选中“Save As”选项单击，弹出如下图所示的界面，在“文件名”栏右侧的编辑框中，键入欲使用的文件名，同时必须键入正确的扩展名。注意，如果用C语言编写程序，则扩展名为(.C)；如果用汇编语言编写程序，则扩展名必须为(.ASM)，扩展名不分大小写。然后，单击“保存”按钮。



（8）将应用程序添加到项目中：单击“Target 1”前面的“＋”号，然后在“Source Group 1”上单击右键，弹出如下菜单

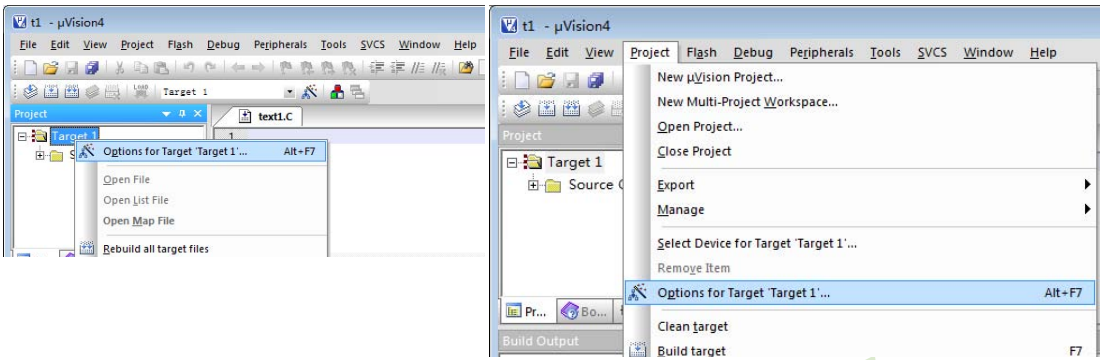


然后单击“Add File to Group ‘Source Group 1’ ”，弹出如下图所示的界面

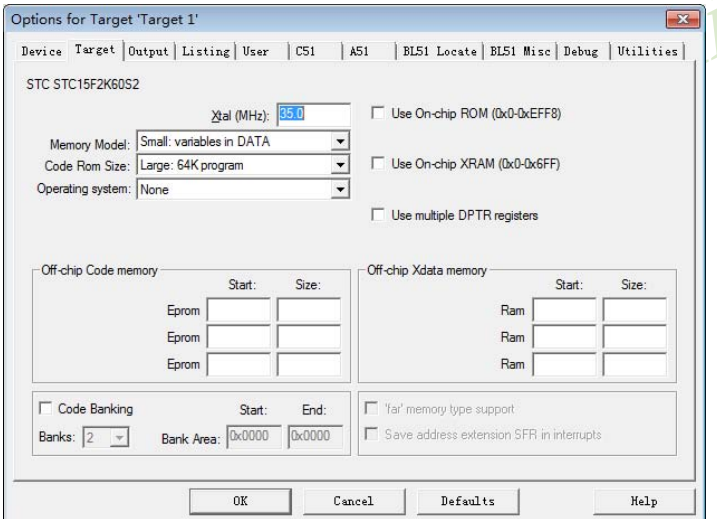


选中text1.c，然后单击“Add”添加成功。

（9）环境设置：在"Target 1"上单击右键选择Options for Target 'Target1'或选择菜单命令Project → Options for Target 'Target1'，弹出Options for Target 'Target1'对话框。

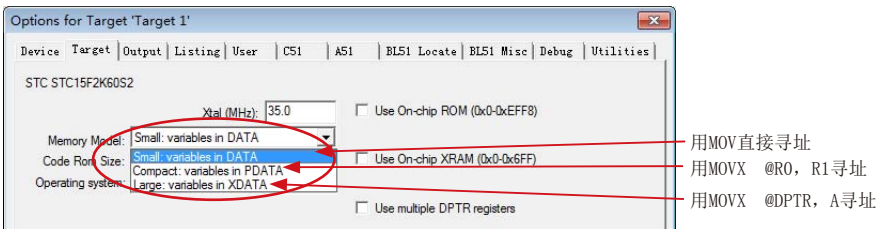


使用Options for Target 'Target1'对话框设定目标的硬件环境。

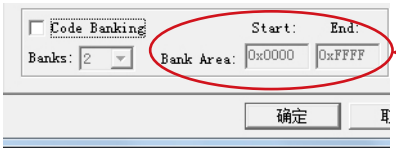


Options for Target 'Target1'对话框有多个选项页，用于设备(Device)选择、目标(Target)属性、输出(Output)属性、C51编译器属性、A51编译器属性、BL51连接器属性、调试(Debug)属性等信息的设置。一般情况下按缺省设置，下面介绍几个需用户自己设置的选项。

① 数据存储器的选择

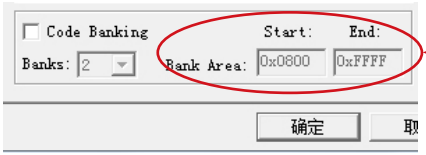


② 程序代码区的起始地址和结束地址默认如下图所示，默认的起始地址或结束地址是合法的。



合法的起始地址和结束地址

但下图的起始地址或结束地址是不合法的，用户须将其修改成为合法的起始地址和结束地址。

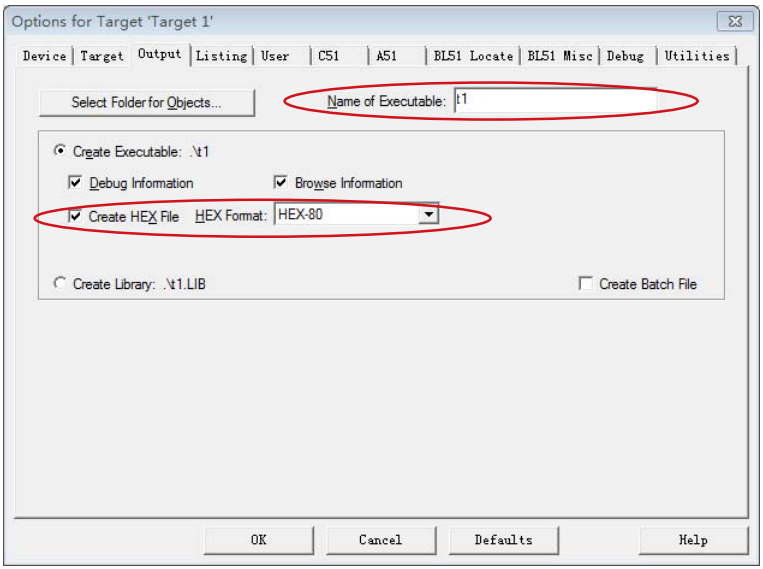


非法的起始地址和结束地址

具体做法如下：先勾选“Code Banking”选项，然后修改“Bank Area”的起始地址和结束地址，最后去选“Code Banking”选项(记住一定要去选此项)，点击【确定】，这样程序代码区的起始地址和结束地址就设置好了。

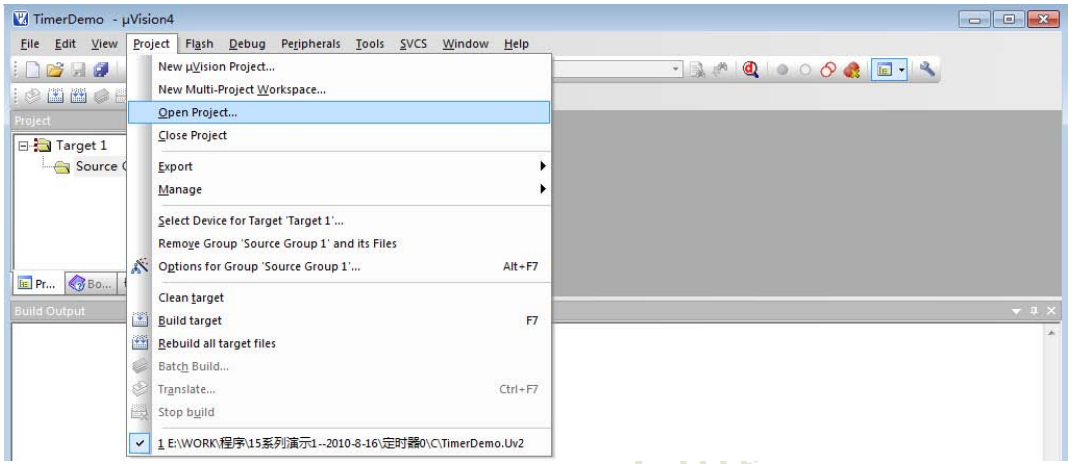
③ 设置在编译、连接程序时自动生成机器代码文件(.HEX)，一定要设置此项，因为默认是不输出HEX代码的，所以需用户设置。

单击“Output”中选项，在弹出的Output对话框中勾选“Create HEX File”选项(如下图所示)，使程序编译后产生HEX代码文件(默认文件名为项目文件名，也可以在“Name of Executable”信息框中输入HEX文件的文件名)，点击【确定】按钮结束设置。

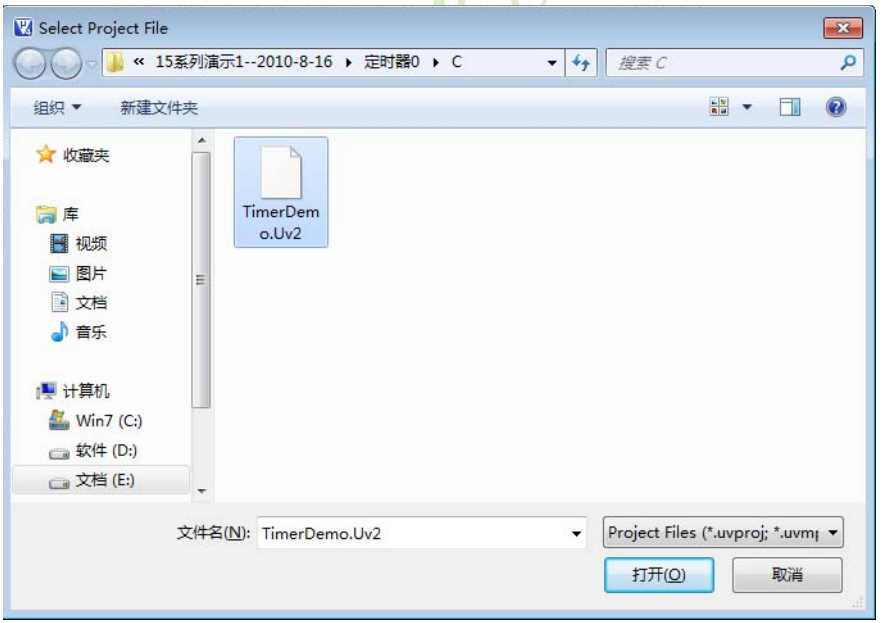


二、如何在用户已建好的项目中改选STC型号MCU进行编译、调试用户程序：

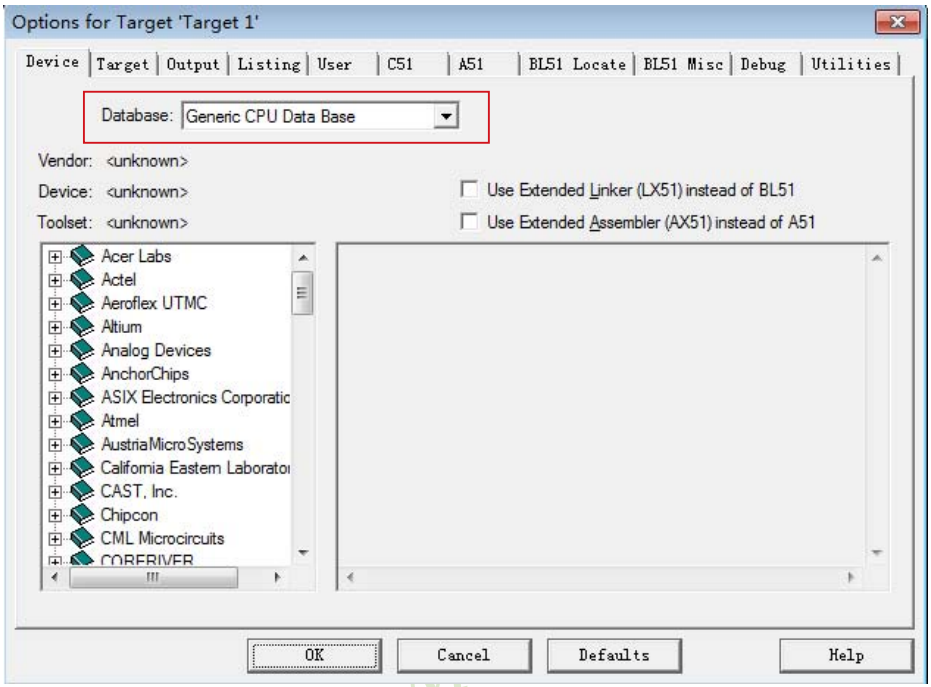
（1）启动Keil μVision4，并打开已建好的项目，如下图所示：



（2）启动Keil μVision4，并打开已建好的项目，在弹出的对话框“Select Project File”中选择目标项目文件，点击【打开】，如下图所示：

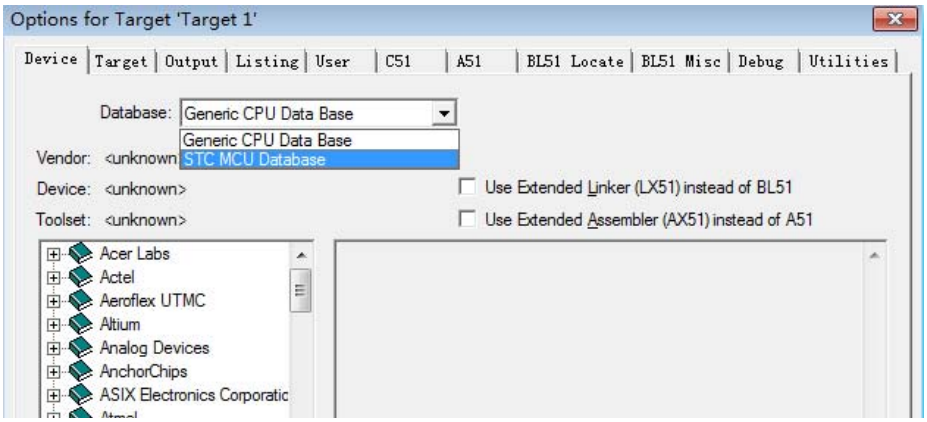


（2）在″Target 1"上单击右键选择Options for Target 'Target1'或选择菜单命令Project→ Options for Target 'Target1'，弹出Options for Target 'Target1'对话框，选择该对话框中“Device”页面，如下图所示：

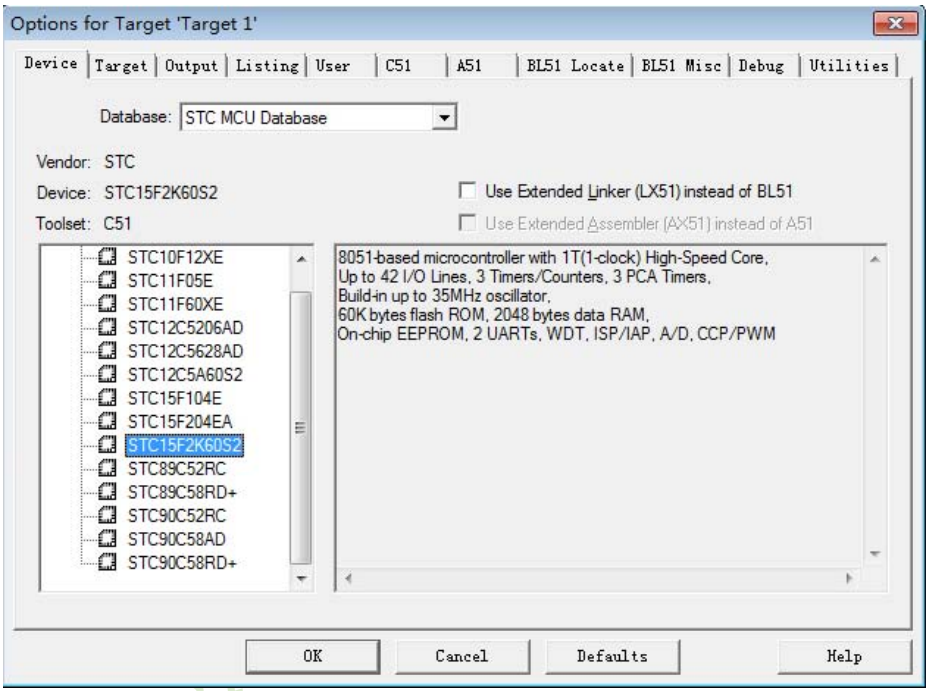


可以看到此时所使用的设备数据库为“通用CPU数据库(Generic CPU Database)”，如用户所使用的单片机为STC单片机，则需更改所使用的设备数据库，具体操作见以下步骤。

（3）因之前已经通过STC-ISP下载编程工具将STC型号MCU添加到Keil μVision4的设备库中(添加方法见上文)，所以此时“Device”页面的中“Database(数据库)”有两个下拉选项“通用CPU数据库(Generic CPU Database)”和“STC MCU数据库(STC MCU Database)”，如下图所示。



在下拉选项中选择“STC MCU数据库(STC MCU Database)”，确定后用户可在左下侧的设备列表选择自己所使用的具体单片机型号，如下图所示。



这样就成功地在已建好的项目中将原MCU改选成了STC型号MCU，接来接用户就可以进行编译、调试用户程序了。

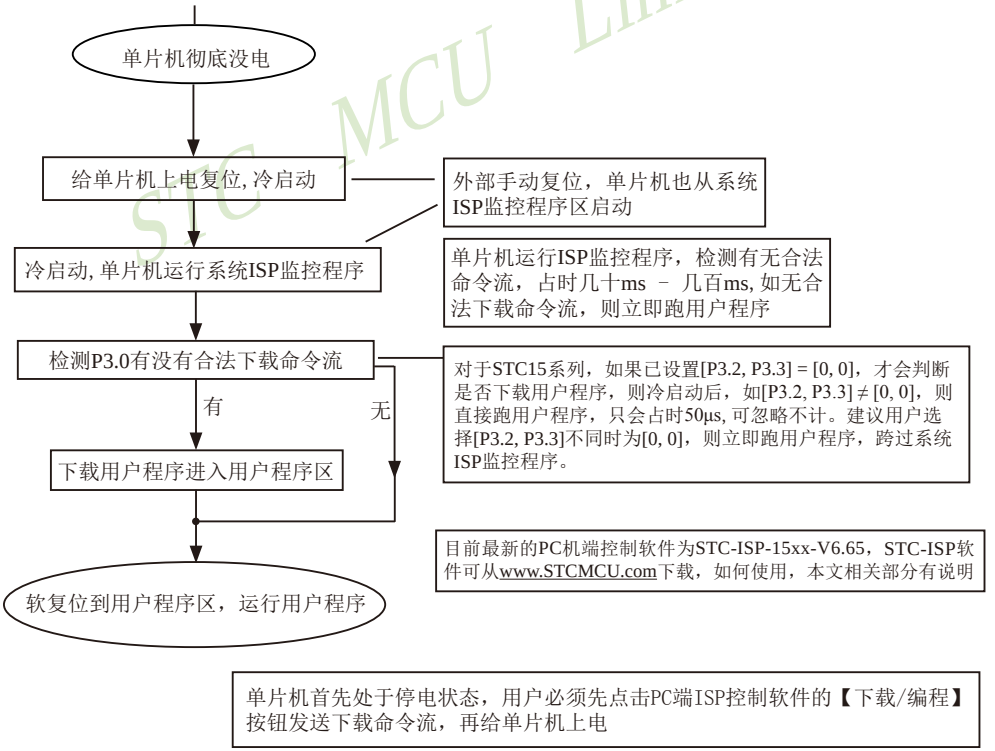
15.2 ISP编程器/烧录器的说明

我们有：STC-ISP经济型下载编程工具

所有STC-ISP编程工具的分类如下：

- STC-ISP经济型下载编程工具
  - STC12/11/10/89/90系列专用ISP下载编程工具
    - STC12/11/10/89/90系列40-pin单片机专用ISP下载编程工具
    - STC12/11/10/89/90系列32-pin单片机专用ISP下载编程工具
    - STC12/11/10/89/90系列28-pin单片机专用ISP下载编程工具
    - STC12/11/10/89/90系列20-pin单片机专用ISP下载编程工具
    - STC12/11/10/89/90系列18-pin单片机专用ISP下载编程工具
    - STC12/11/10/89/90系列16-pin单片机专用ISP下载编程工具
  - STC15系列专用ISP下载编程工具

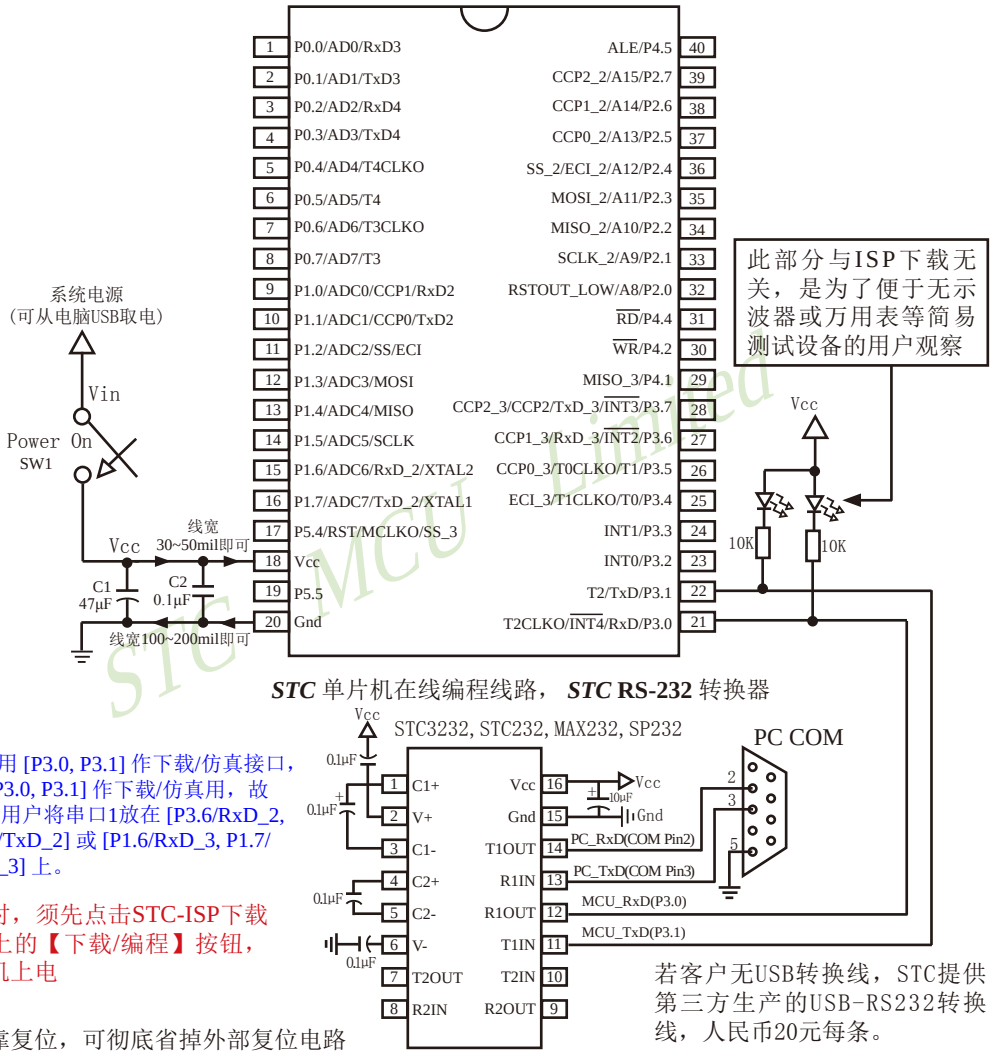
15.2.1 在系统可编程(ISP)原理使用说明



STC MCU Limited

15.2.2 STC15系列在系统可编程(ISP)典型应用线路图

15.2.2.1 利用RS-232转换器的ISP下载典型应用线路图



注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口1放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

烧录程序时，须先点击STC-ISP下载编程工具上的【下载/编程】按钮，再给单片机上电

若客户无USB转换线，STC提供第三方生产的USB-RS232转换线，人民币20元每条。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚.

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

如何产生虚拟串口:①安装Windows驱动程序；②插上USB-RS232转换线(若客户无USB转换线，STC提供第三方生产的USB-RS232转换线，人民币20元每条.)；③确定PC端口COM：右击我的电脑—>属性—>硬件—>设备管理器—>确定所扩展的串口是PC电脑虚拟的第几个COM.

STC系列单片机具有在系统可编程(ISP)特性，ISP的好处是：省去购买通用编程器，

单片机在用户系统上即可下载/烧录用户程序，而无须将单片机从已生产好的产品上拆下，再用通用编程器将程序代码烧录进单片机内部。有些程序尚未定型的产品可以一边生产，一边完善，加快了产品进入市场的速度，减小了新产品由于软件缺陷带来的风险。由于可以在用户的目标系统上将程序直接下载进单片机看运行结果对错，故无须仿真器。

STC系列单片机内部固化有ISP系统引导固件，配合PC端的控制程序即可将用户的程序代码下载进单片机内部，故无须编程器(速度比通用编程器快，几秒一片)。

如何获得及使用STC提供的ISP下载工具(STC-ISP.exe软件)：

(1). 获得STC提供的ISP下载工具(软件)

登陆 www.STCMCU.com 网站，从STC半导体专栏下载PC(电脑)端的ISP下载工具(软件)，然后将其自解压，再安装即可(执行setup.exe)，注意随时更新软件。

(2). 使用STC-ISP下载工具(软件)，请随时更新，STC-ISP下载工具目前已到Ver6.57版本。

支持*.bin,*.hex(Intel 16 进制格式)文件，少数*.hex文件不支持的话，请转换成*.bin文件，请随时注意升级PC(电脑)端的STC-ISP.exe软件。

(3). STC系列单片机出厂时就已完全加密。需要单片机内部的电放光后上电复位(冷起动)才运行系统ISP监控程序，如从P3.0检测到合法的下载命令流就下载用户程序，如检测不到就复位到用户程序区，运行用户程序。

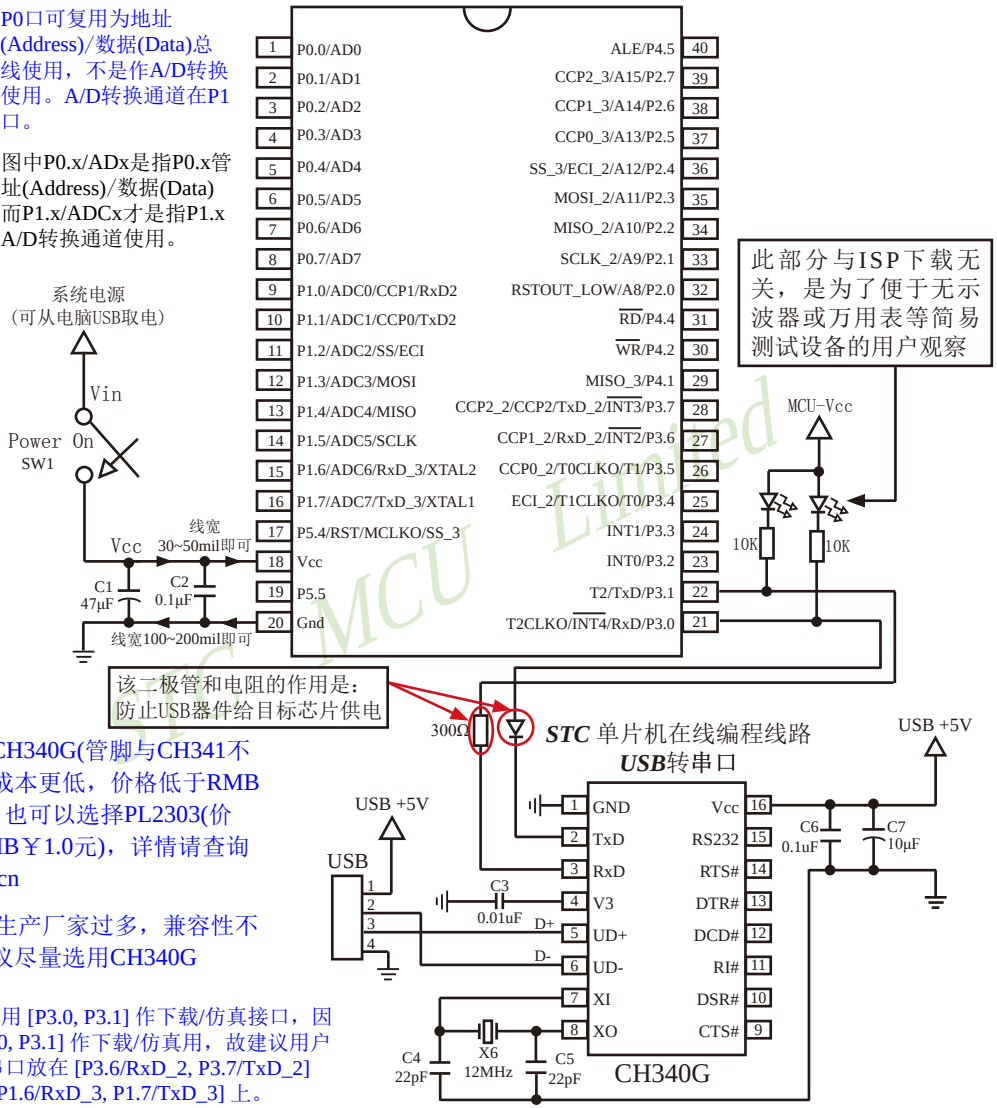
(4). 如果用户系统接了RS-485通信电路，推荐将RS-485电路接到 [P1.6, P1.7] 或 [P3.6, P3.7] 上，这样既方便又安全，且不用在STC-ISP下载编程工具中选择“下次冷启动时需 [P3.2,P3.3] = [0, 0]才可以下载程序”。

STC MCU Limited

15.2.2.2 利用USB转串口的ISP下载典型应用线路图

特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路

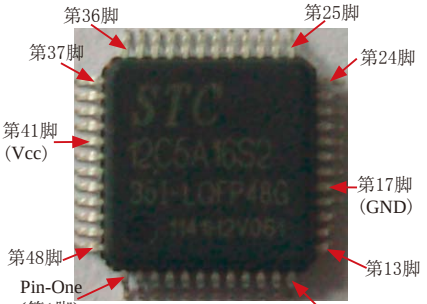
P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

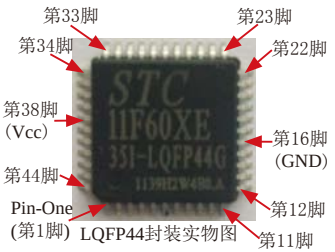
建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

15.2.3 所有STC系列单片机封装实物图

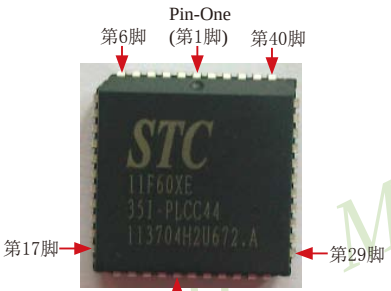
STC12/11/10/89/90系列单片机的封装实物图:



LQFP48封装实物图

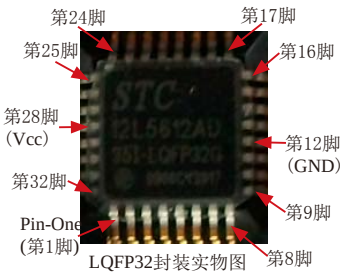


LQFP44封装实物图



PLCC44封装实物图

40-Pin单片机PDIP40的封装实物图



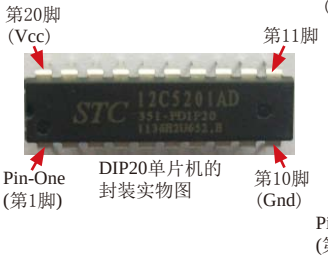
LQFP32封装实物图

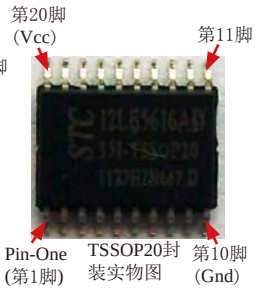
SOP32封装实物图

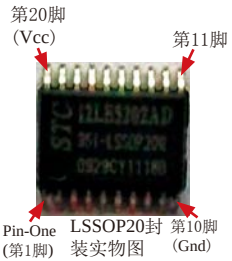
SOP28封装实物图

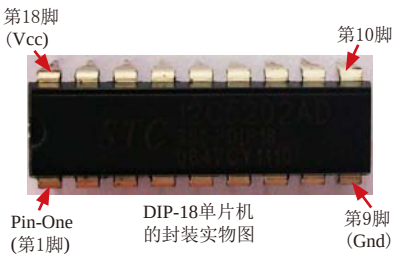
SKDIP28单片机的封装实物图

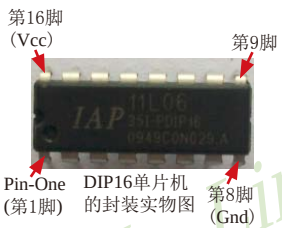
STC MCU Limited

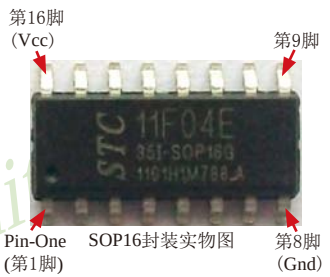




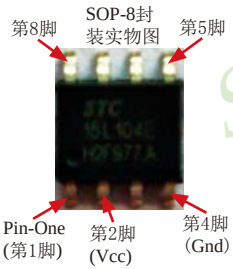


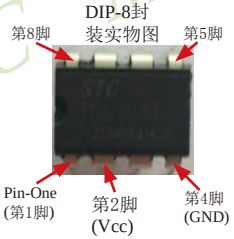


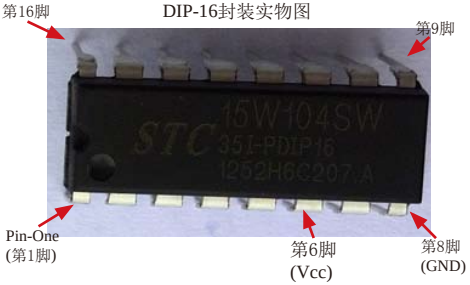




STC15系列单片机的封装实物图：

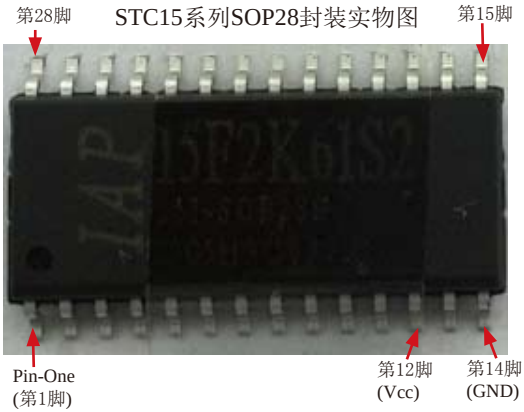






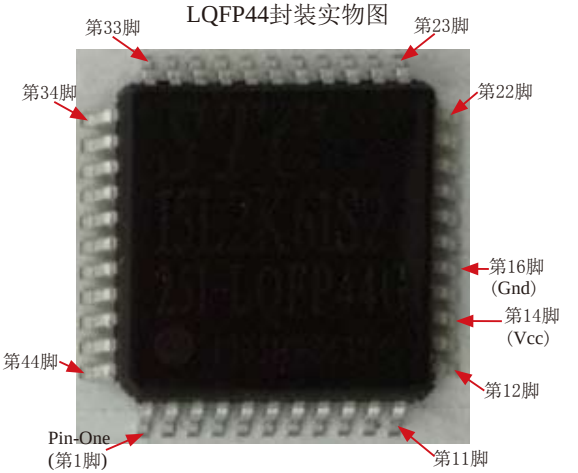
STC MCU Limi

STC15系列SOP28封装实物图



STC15系列PDIP40封装实物图

LQFP44封装实物图



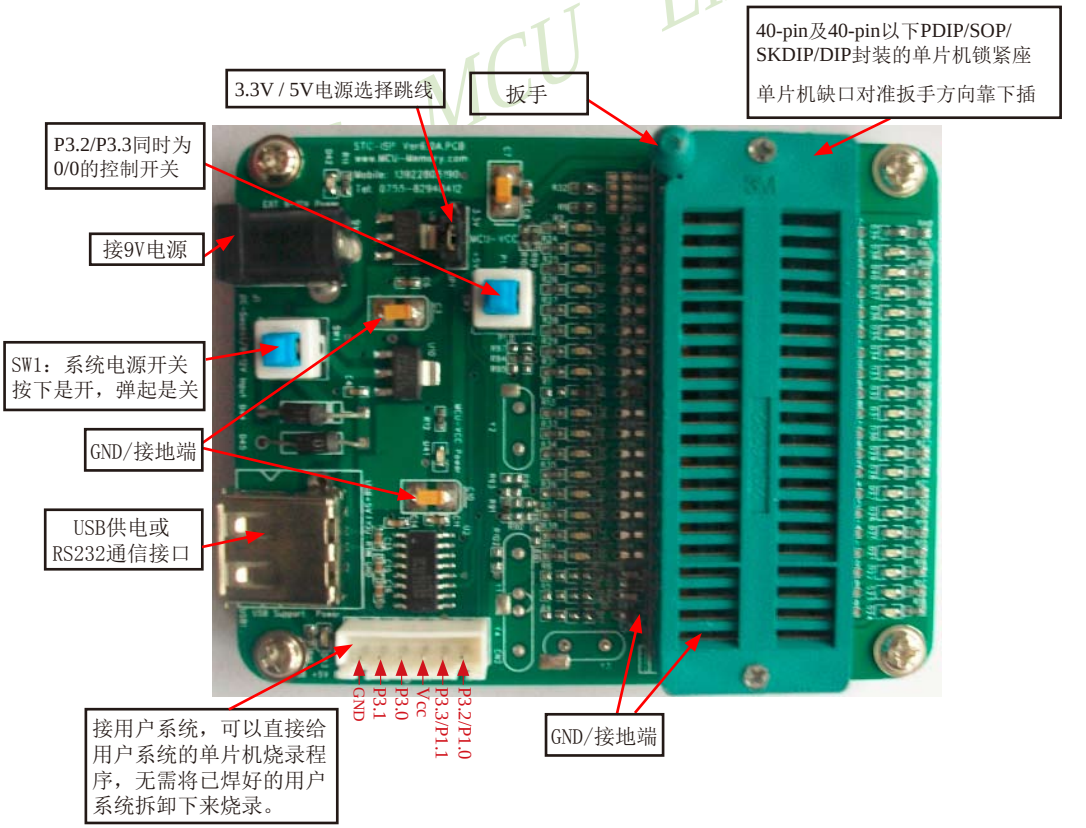
15.2.4 STC-ISP下载编程工具硬件——STC-ISP下载板

15.2.4.1 STC15系列ISP下载板实物图

STC15系列单片机专用ISP下载编程工具实物图

STC15系列ISP下载编程工具与STC12/11/10/89/90系列的ISP下载编程工具不兼容，因此注意此ISP下载编程工具适用的单片机型号

STC15系列专用ISP下载编程工具为例详细介绍STC-ISP下载板的布局：

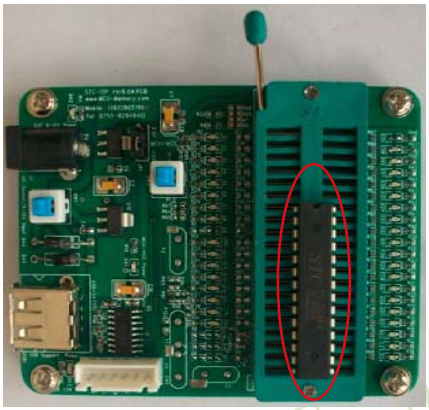


15.2.4.2 如何将单片机安装到STC-ISP下载板上

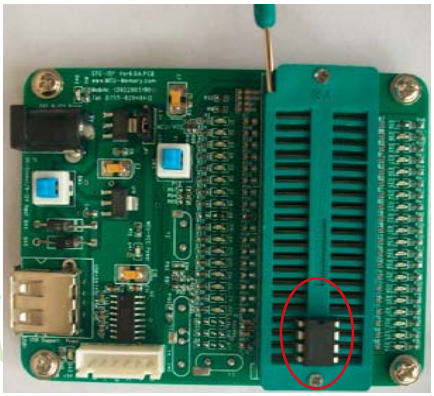
根据用户所使用的单片机型号及管脚选择相应的STC-ISP下载板，先将下载板上的扳手向上弹起，然后将单片机插入相应的STC-ISP下载板的锁紧座上(具体做法是：将芯片的半圆缺口对准扳手的方向靠下插)，最后将扳手向下按锁紧单片机。

注意：不管是哪种STC-ISP下载编程工具，其正面焊的编程烧录用锁紧座都是40Pin的，**锁紧座第20-Pin接的是地线(GND)，所以请将单片机的地线对着锁紧座的地线插即**将芯片的半圆缺口对准扳手的方向靠下插。

STC15系列SKDIP28封装的插法



STC15系列DIP-8封装的插法



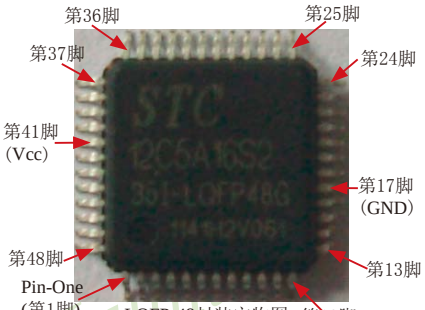
STC MCU

15.2.4.3 如何使用转换座将贴片封装的单片机安装到STC-ISP下载板上

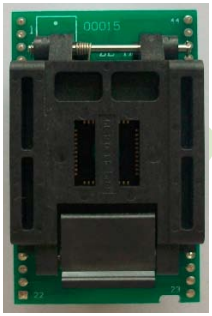
STC-ISP下载板的编程烧录锁紧座只能插入40 Pin及40 Pin以下的直插式的单片机，对于LQFP、PLCC、SOP等封装的单片机需转换座将这些封装转换成直插式的封装才能插入STC-ISP下载板中。下面介绍几种常用的转换座以及如何使用这些转换座。

LQFP-48的转换座(正面)

LQFP-48的转换座(内部)



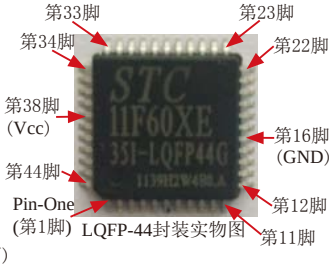
LQFP-48封装实物图



LQFP-44的转换座(正面)

LQFP-44的转换座(内部)

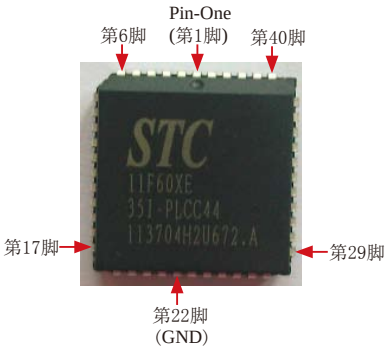
LQFP-44的转换座(反面)



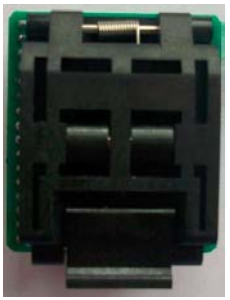
LQFP-44封装实物图

PLCC-44的转换座(正面)

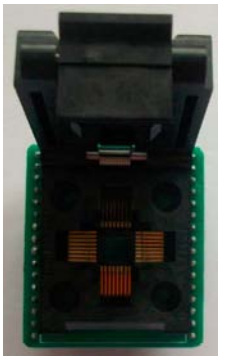
PLCC-44的转换座(反面)



PLCC-44封装实物图

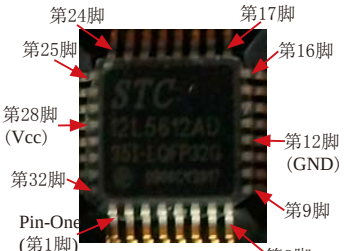


LQFP-32的转换座(正面)

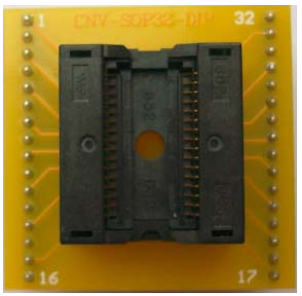


LQFP-32的转换座(内部)

LQFP-32的转换座(反面)



LQFP-32封装实物图



SOP-32的转换座(正面)

SOP-32的转换座(反面)

SOP-32封装实物图

SOP-28和SOP-20
的转换座(正面)

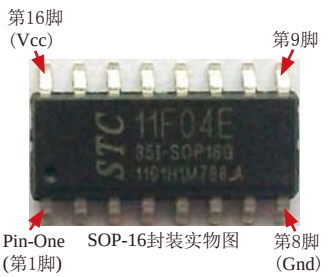
SOP-28和SOP-20
的转换座(反面)

SOP-28封装实物图

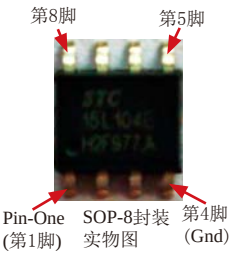
SOP-20封
装实物图

SOP-16和SOP-8
的转换座(正面)

SOP-16和SOP-8
的转换座(反面)



SOP-16封装实物图



SOP-8封装
实物图

给需转换座的单片机烧录程序的具体步骤如下：

（1）根据单片机的封装选择转换座，并将单片机安装进转换座中：

LQFP-48/LQFP-44/LQFP-32封装的单片机按下图所示安装；

① 打开转换座的盖子

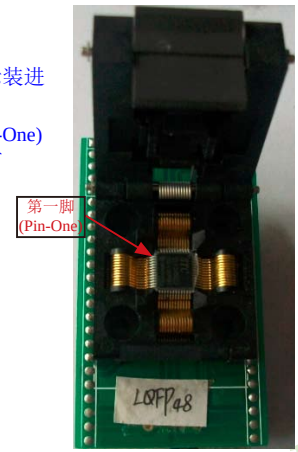
↓

② 将单片机按右图中所示装进转换座中

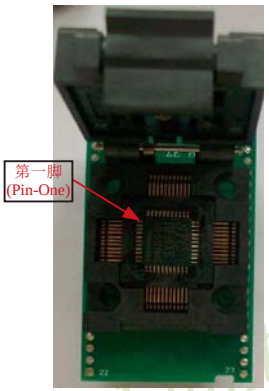
注意：单片机的第一脚(Pin-One)对准转换座的左上方

↓

③ 盖上转换座的盖子



LQFP-48封装的单片机安装图

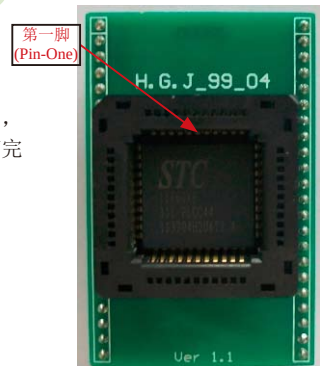


LQFP-44封装的单片机安装图

LQFP-32封装的单片机安装图

PLCC-44封装的单片机按下图所示安装；

首先将单片机正对准转换座上插槽(按右图所示对准)，然后平稳地将单片机推进转换座的插槽中，直到插槽完全嵌牢了单片机。



PLCC-44封装的单片机安装图

SOP-32和SOP-28/20以及SOP-16/8封装的单片机按下图所示安装；

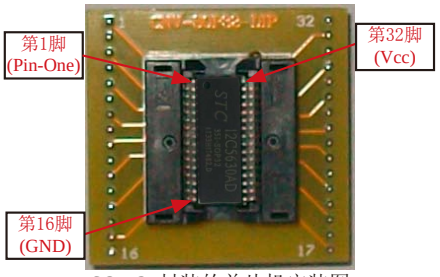
① 将转换座上安有弹簧的左右两边往下按

↓

② 将单片机正对准转换座并靠下插入插槽中(按右图所示对准)

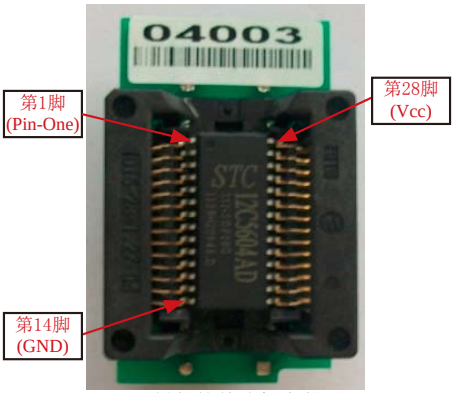
↓

③ 松开转换座上安有弹簧的左右两边

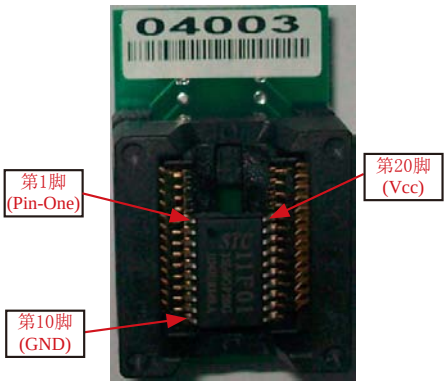


SOP-32封装的单片机安装图

STC MCU Limited

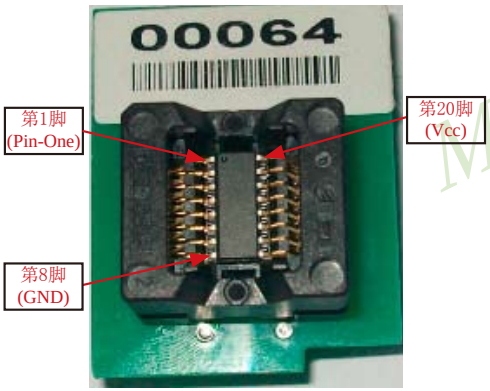


SOP-28封装的单片机安装图

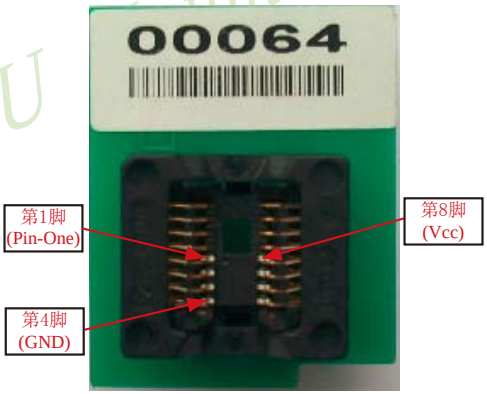


SOP-20封装的单片机安装图

SOP-28和SOP-20封装的单片机用同一个转换座(SOP-28转换座)，将单片机正对准转换座并靠下插入转换座的插槽中。



SOP-16封装的单片机安装图

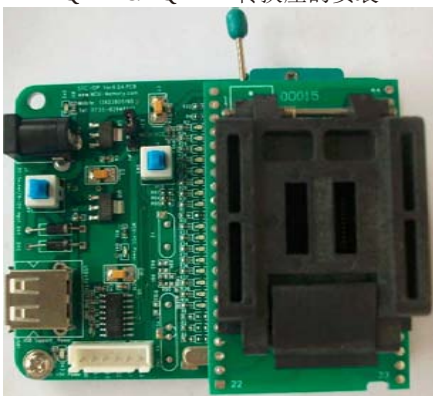


SOP-8封装的单片机安装图

SOP-16和SOP-8封装的单片机用同一个转换座(SOP-16转换座)，将单片机正对准转换座并靠下插入转换座的插槽中。

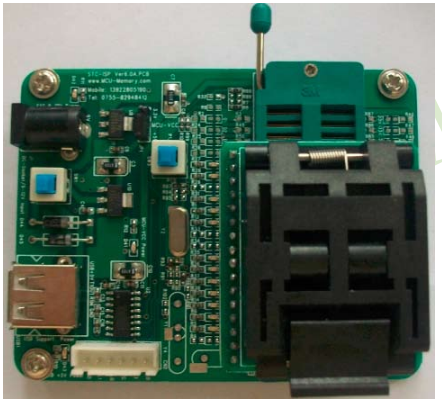
（2）将安有单片机的转换座安装在与单片机相对应的STC-ISP下载板锁紧座上，具体做法是：将转换座正对准扳手的方向靠下插。

LQFP-48/LQFP-44转换座的安装

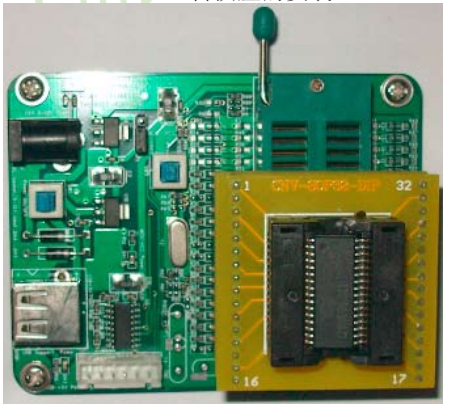


PLCC-44转换座的安装

LQFP-32转换座的安装



SOP-32转换座的安装

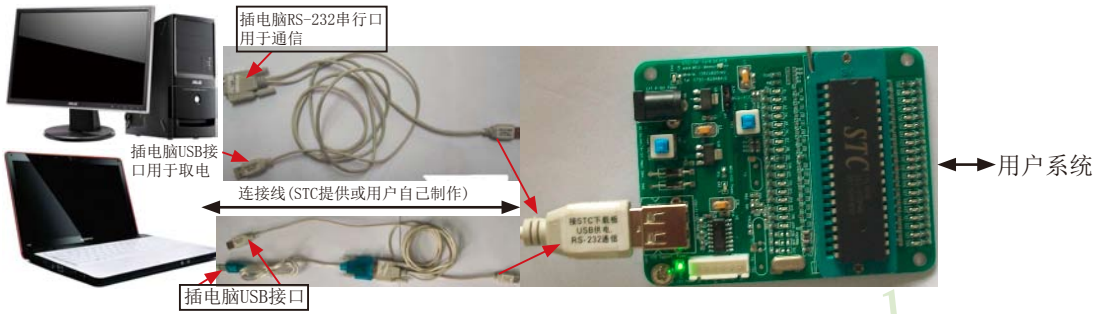


SOP-28转换座的安装

SOP-16转换座的安装

15.2.4.3 如何将STC-ISP下载板连接到电脑

STC-ISP下载编程工具其实就是单片机通过RS-232转换器连接到电脑完成下载编程用户程序工作的。



有些笔记本电脑没有标准RS-232串行口，需一条USB-RS232转换线来扩展一个RS-232串行口。市场上有很多种USB-RS232转换线，有的是不能与STC下载板或电脑操作系统兼容的。请尽量选择用CH340/CH341做的USB-RS232转换线或让STC帮你购买经过测试的转换线。如果是用PL2303或CP2102制作的USB-RS232转换线，请尝试安装不同版本的驱动程序解决它们的不兼容问题。

关于硬件连接：

(1). MCU/单片机　RXD(P3.0)　---　RS-232转换器　---　电脑　TXD(COM Port Pin3)

(2). MCU/单片机　TXD(P3.1)　---　RS-232转换器　---　电脑　RXD(COM Port Pin2)

(3). MCU/单片机　GND　----------　电脑　GND(COM Port Pin5)

(4). 如果您的系统接了RS-485通信电路，推荐将RS-485电路接到 [P1.6, P1.7] 或 [P3.6, P3.7] 上，这样既方便又安全，且不用在STC-ISP下载编程工具中选择“下次冷启动时需[P3.2, P3.3] = [0, 0]才可以下载程序”。

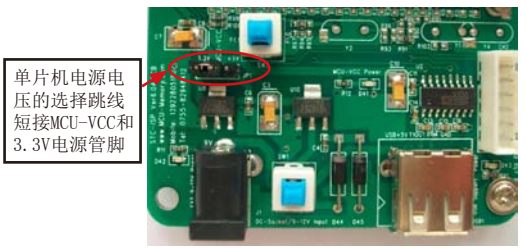
(5). RS-232转换器可选用MAX232/SP232(4.5-5.5V),MAX3232/SP3232(3V-5.5V).

STC-ISP下载板连接电脑的具体方式:

(1).根据单片机的工作电压在STC-ISP下载板上选择单片机电源电压

A). 5V单片机，将MCU-VCC和+5V电源管脚短接

B). 3V单片机，将MCU-VCC和3.3V电源管脚短接

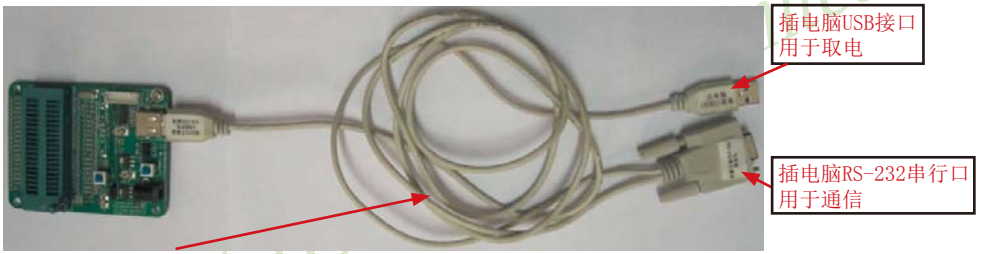


(2).将STC-ISP下载板连接到电脑端

根据用户所使用的电脑是否有RS-232串行口选择连接电脑的方式。

A).如果用户电脑有RS-232串行口，参照下图连接。

下面是STC-ISP下载板连接有RS-232串行口电脑的方式:



连接线(STC提供或用户自己制作)的连接方法:

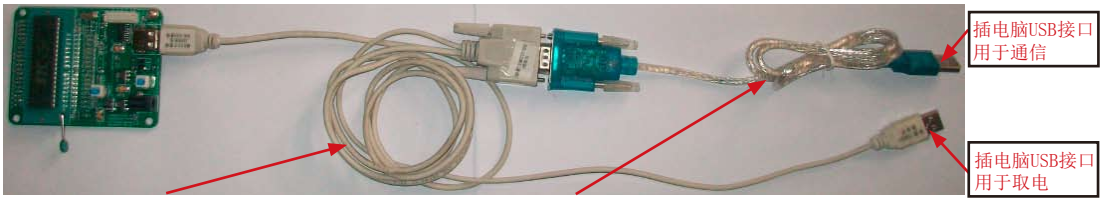
①.将一端有9芯连接座的插头插入电脑RS-232串行接口插座用于通信;

②.将连接线的“从电脑USB口取电”的USB插头插入电脑USB接口用于取电;

③.将连接线中“接STC下载板”的USB插头插入STC-ISP下载编程工具的PCB板USB1插座用于RS-232通信和供电

B).如果用户电脑没有RS-232串行口，参照下图连接。

下面是STC-ISP下载板连接没有RS-232串行口电脑(需一条USB-RS232转换线扩展一个RS232串行口)的方式:



连接线(STC提供或用户自己制作)和USB-RS232转换线的连接方法:

①.将连接线中一端有9芯连接座的插头插入USB-RS232转换线的相应插座中;

②.将连接线的“从电脑USB口取电”的USB插头插入电脑USB接口用于取电;

③.将USB-RS232转换线中的USB插头插入电脑USB接口用于通信

④.将连接线中“接STC下载板”的USB插头插入STC-ISP下载编程工具的PCB板USB1插座用于RS-232通信和供电

(3).其他插座不需连接

(4).“系统电源开关Power ON”开关处于非按下状态，此时MCU-VCC Power灯不亮,没有给单片机通电

(5).通过“[P3.2, P3.3] = [0,0] (对于STC12系列、STC11系列、STC10系列、STC89系列及STC90系列为[P1.0, P1.1] = [0,0]) ”控制开关：

处于非按下状态，[P3.2, P3.3] = [1, 1] , 不短接到地；

处于按下状态，　[P3.2, P3.3] = [0, 0] , 短接到地。

如果单片机已被设成“下次冷启动[P3.2, P3.3] = [0, 0] 才判P3.0有无合法下载命令流”就必须将此开关处于按下状态，让单片机的[P3.2, P3.3]短接到地

(6).将单片机插进锁紧座，锁紧单片机,注意单片机是8-Pin/20-Pin/28-Pin/32-Pin/40-Pin的，锁紧座是40-Pin,我们的设计是靠下插，单片机地线(Gnd)对准锁紧座的地线(Gnd)插。

STC MCU Limited

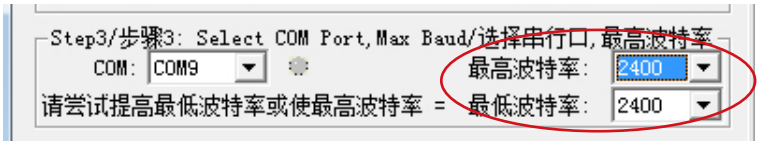
15.2.5 针对USB-RS232转换线不兼容问题的几点说明

有些新式笔记本电脑没有标准RS-232串行口，则需要一条USB-RS232转换线来扩展一个RS-232串行口。但有些USB-RS232转换线与STC下载板或电脑操作系统是不能兼容的，这里针对这些不兼容问题提出几点解决方法：

（1）请尽量选择用CH340/CH341制作的USB-RS232转换线

（2）对于市场上有些用PL2303或CP2102制作的USB-RS232转换线，尝试安装不同版本的驱动程序解决它的不兼容问题。

（3）尝试在STC-ISP控制下载软件中将最高波特率和最低波特率设置为相等且都为2400，重新连接。



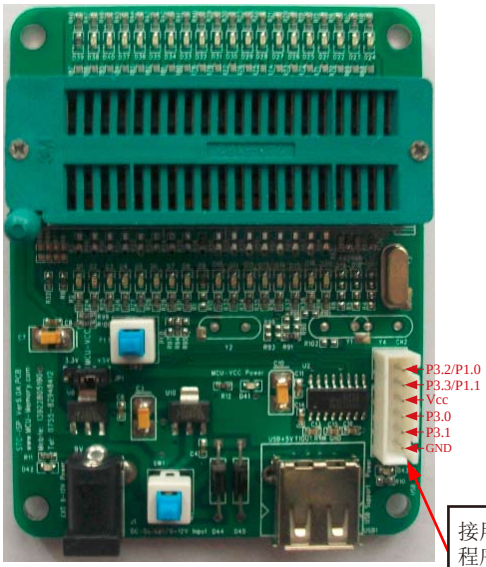
（4）让STC帮您购买经过测试的转换线。

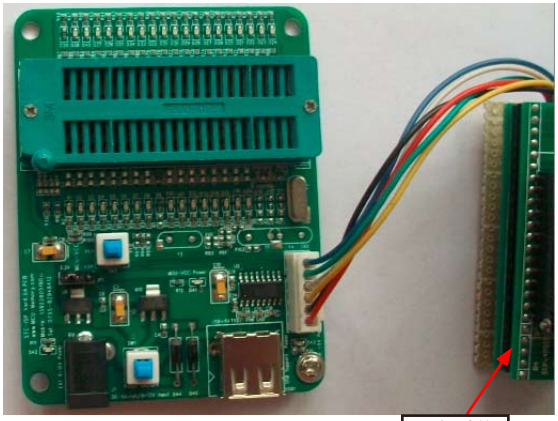
15.2.6 如何用STC-ISP下载板给在用户系统上的单片机烧录用户程序

利用STC系列ISP下载编程工具(其实就是单片机通过RS-232转换器连接到电脑)进行RS-232转换。

单片机在用户自己的板上完成下载/烧录：

1. U1-Socket锁紧座不得插入单片机
2. 将用户系统上的电源(MCU-VCC,GND)及单片机的[P3.0, P3.1]接入转换板的“白色六芯插座”，如下图所示，这样用户系统上的单片机就具备了与电脑进行通信的能力
3. 将用户系统的单片机的[P3.2, P3.3] (对于STC12系列、STC11系列、STC10系列、STC89系列及STC90系列为[P1.0, P1.1])接入转换板“白色六芯插座”(如果需要的话)
4. 如须 [P3.2, P3.3] = [0, 0], 短接到地，可在用户系统上将其短接到地，或将 [P3.2, P3.3] 也从用户系统引到STC系列ISP下载编程工具 (其实就是单片机通过RS-232转换器连接到电脑)上，将“控制[P3.2, P3.3] 同时为[0, 0]的开关”按下，则[P3.2, P3.3] = [0, 0]。
5. 将STC-ISP下载板连接到电脑上进行RS232通信(具体连接方式见下页图)
6. 给单片机上电复位(注意是从用户系统自供电，不要从电脑USB取电, 电脑USB座不插)
7. 关于软件：选择“Download/下载”
8. 下载程序时，如用户板有外部看门狗电路，不得启动，单片机必须有正确的复位, 但不能在ISP下载程序时被外部看门狗复位，如有，可将外部看门狗电路WDI端或WDO端浮空。
9. 如系统接了RS-485通信电路，推荐将RS-485电路接到[P1.6, P1.7]或[P3.6, P3.7]上，这样既方便又安全，且不用在STC-ISP下载编程工具中选择“下次冷启动时需[P3.2, P3.3] = [0, 0]才可以下载程序”

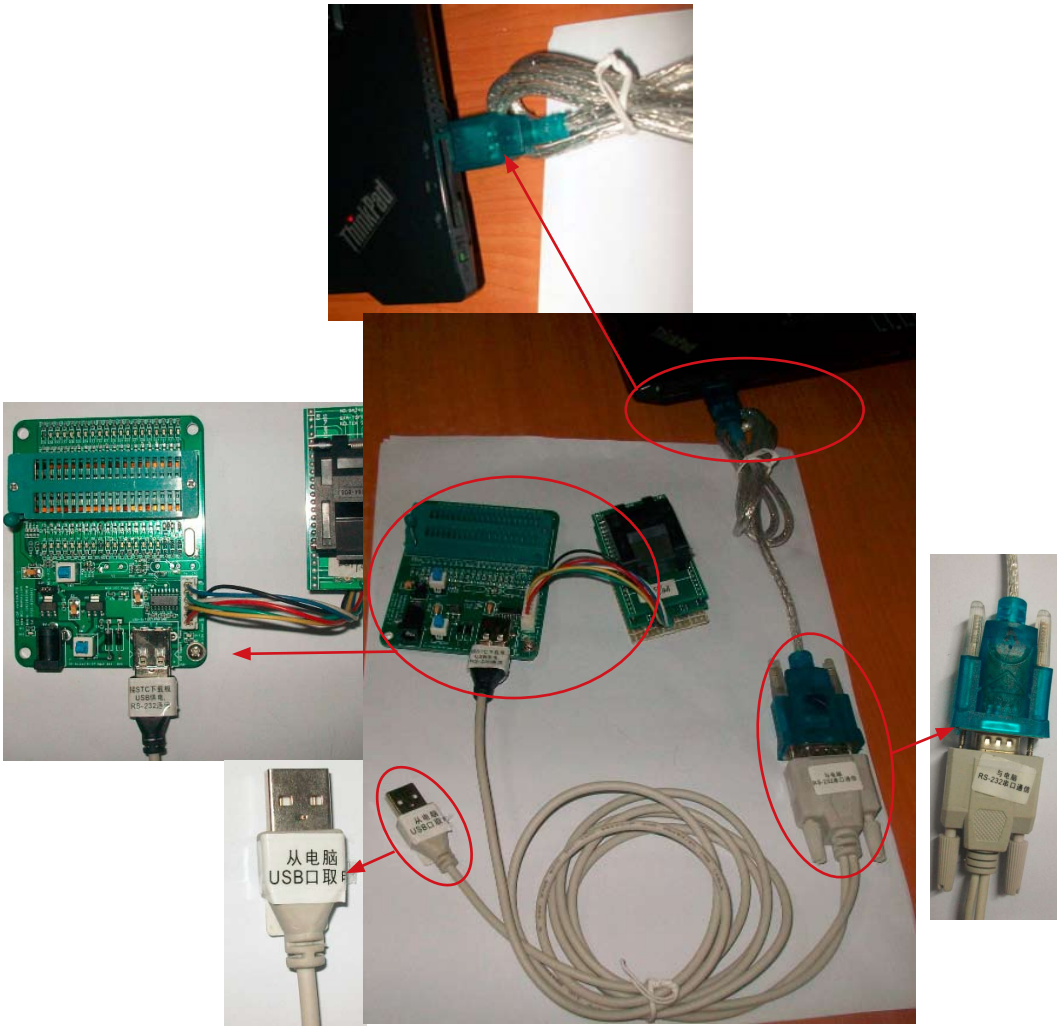




将用户系统按上图所示连接到STC-ISP下载板上

接用户系统，可以直接给用户系统的单片机烧录程序，无需将已焊好的用户系统拆卸下来烧录。

STC MCU Limited



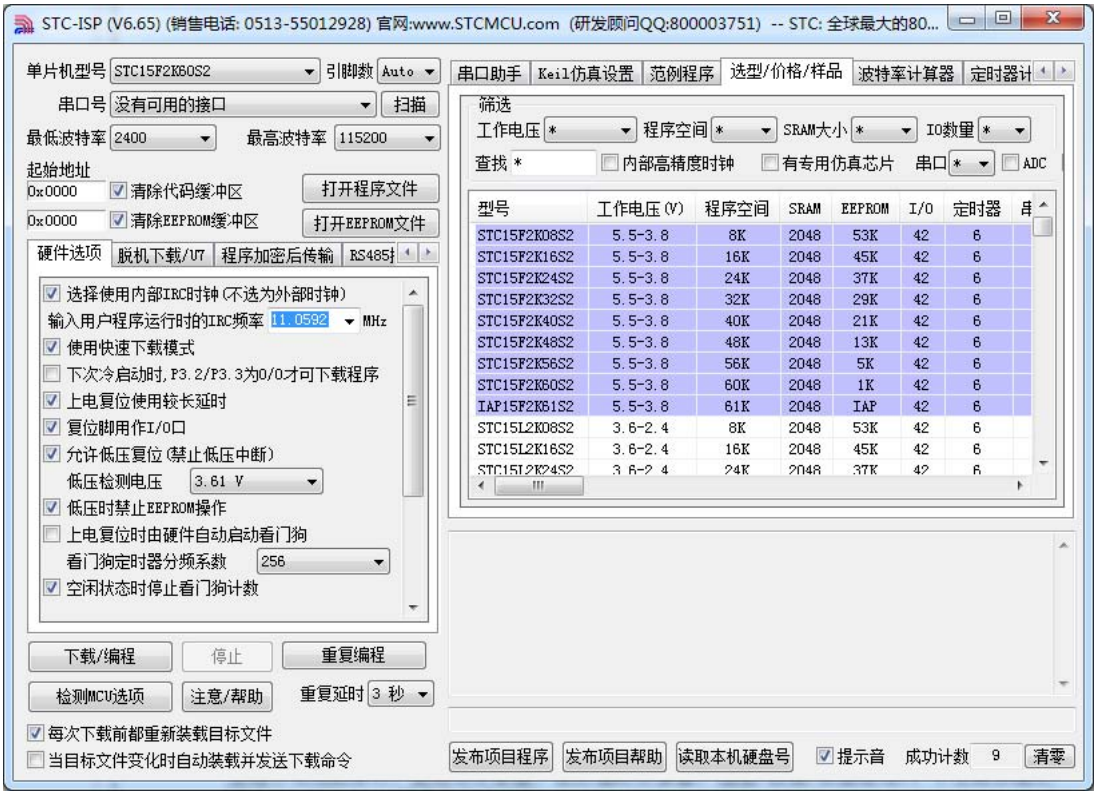
将连有用户系统的STC-ISP下载板按左图所示连接到电脑上，注意以下几点：

（1）STC-ISP下载板的锁紧座不得插入单片机；

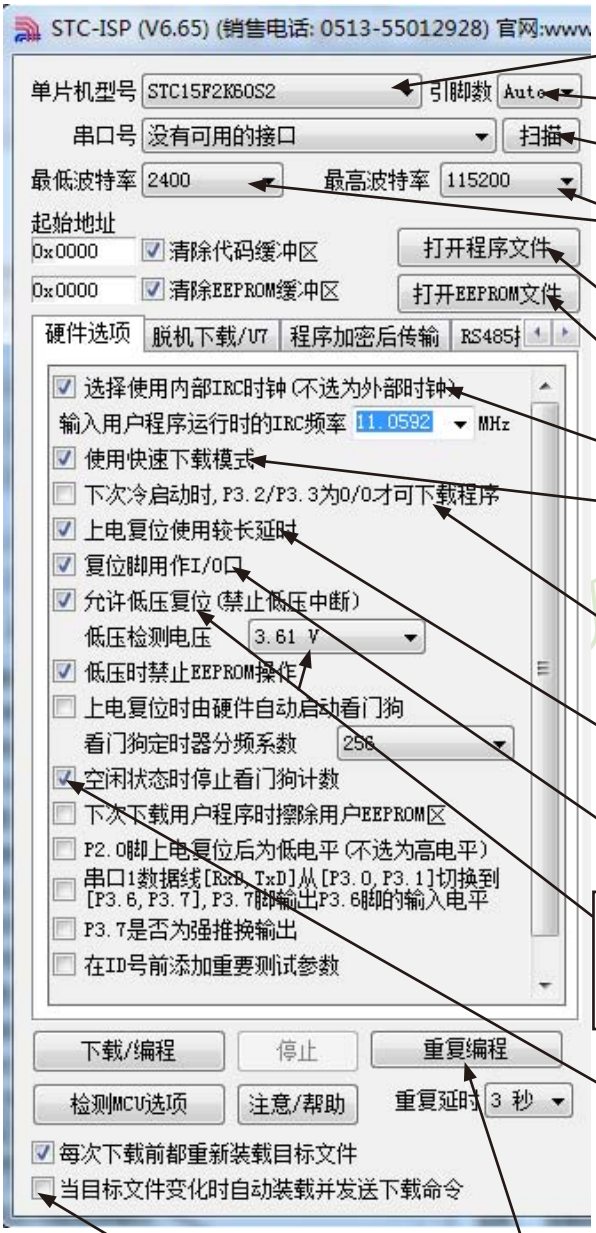
（2）“从电脑USB口取电”的USB插头悬空，不要插入电脑，因为是**从用户系统自供电的。**

（3）接STC下载板的USB插头仅用于RS232通信。

15.2.7 电脑端的STC-ISP控制软件(Ver6.65)的界面使用说明



最新的ISP下载控制软件V6.65的界面如上图所示。该软件新增了许多新功能(如扫描当前系统中可用的串口、波特率计算器、软件延时计算器、选型/价格/样品表等)。下文将详细介绍该STC-ISP-V6.65软件的各个功能。



选择STC系列单片机的型号

选择STC15系列单片机的封装

扫描当前系统中可用的串口

用户根据实际使用效果选择限制最高或最低波特率，如57600, 38400,19200,2400或Auto Buad

打开用户的程序代码文件

打开EEPROM数据文件

选择时钟(内部R/C时钟)频率(可输入)

是否使用较快速度的内部振荡器频率进行下载
选择：　使用较快频率的内部振动器
不选择：使用较慢频率的内部振动器

下次是否需要[P3.2, P3.3]同时为低电平时才可下载程序
选择：　[P3.2, P3.3]同时为低电平时才可下载程序
不选择：下载时不检测[P3.2, P3.3]的电平

上电复位时，是否需要额外的复位延时
选择：　需要额外的复位延时
不选择：一般长度的复位延时

是否需要将复位引脚当作普通I/O口来使用
选择：　复位引脚当作普通I/O口
不选择：复位引脚仍为复位脚

当电压低于设定的低压检测门槛电压时，芯片是复位还是中断
选择：　检测到低压时复位
不选择：检测到低压时不复位而产生低压中断
建议：当振动器频率高于20MHz时
对于3V的芯片，低压检测门槛电压建议选择2.5V以上
对于5V的芯片，低压检测门槛电压建议选择4.11V以上

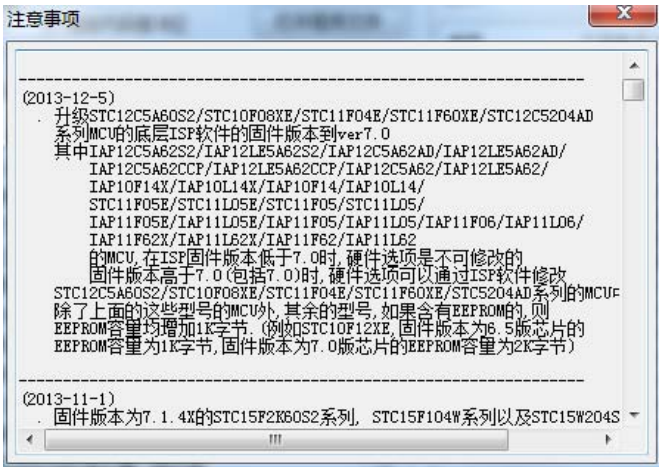
当芯片处于空闲状态时，是否需要停止内部看门狗计数
选择：　空闲状态时停止计数
不选择：空闲状态时继续计数

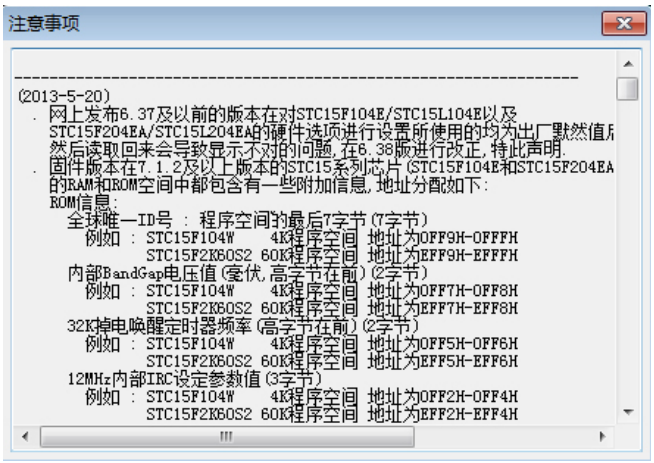
新的设置冷启动后（彻底停电后再上电），才生效

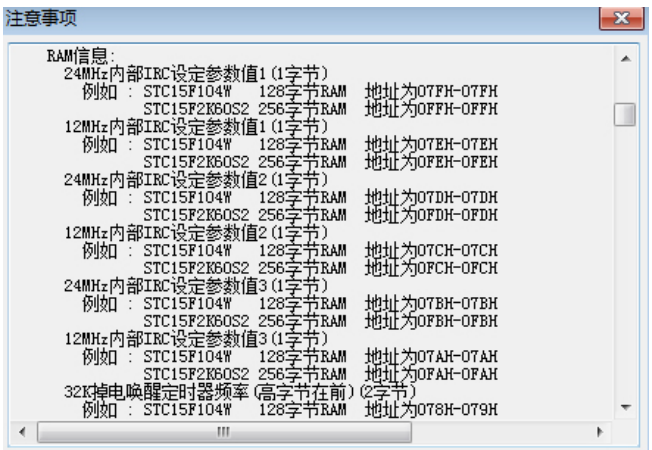
大批量生产时使用

如P3.0/P3.1外接RS-485/RS-232等通信电路，建议选择P3.2/P3.3等于0/0才可以下载程序，如不同时为0/0，则跨过系统ISP引导程序，直接运行用户程序。

点击界面上的注意/帮助按钮后出现下面的对话框：

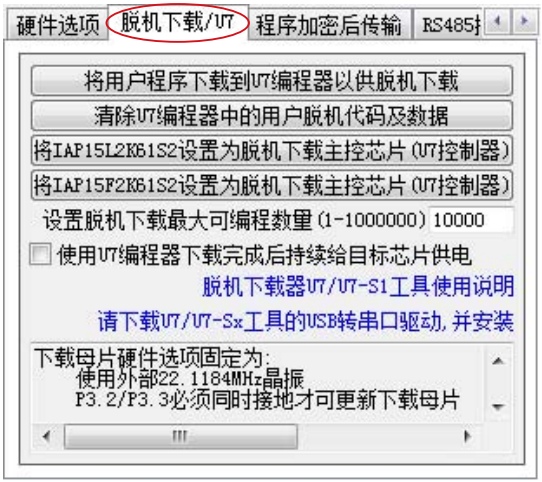




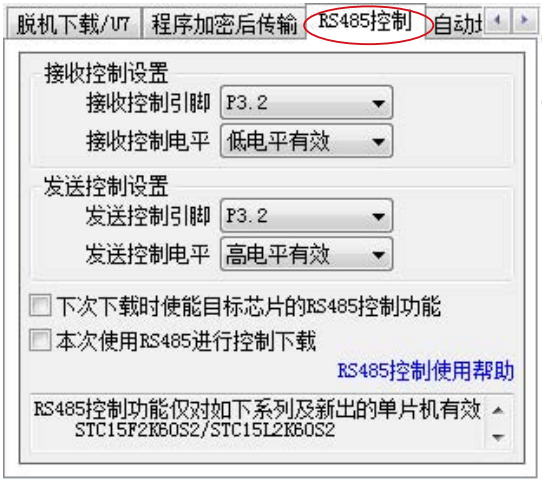


Limited

脱机下载界面：



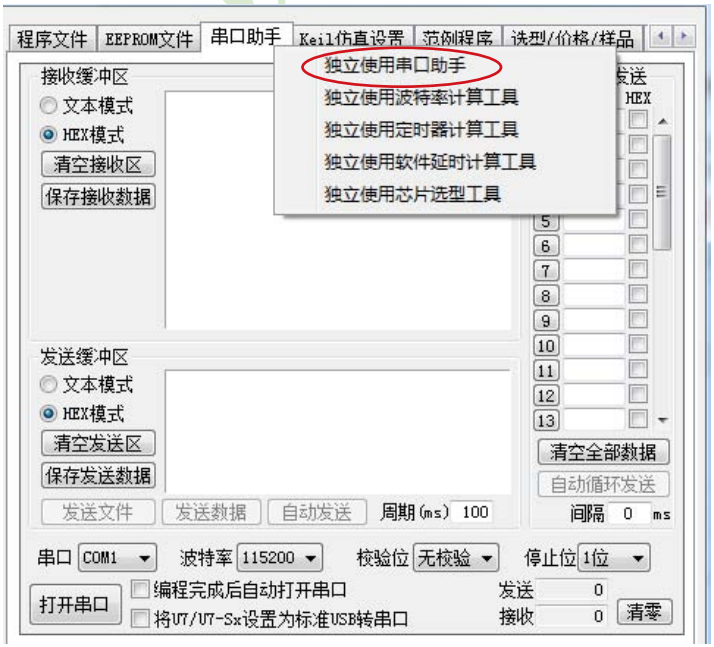
RS485控制界面：



STCMCU Limited

串口助手界面：

在串口助手工具选择页上单击鼠标右键进行选择，可以将串行口助手从STC-ISP下载编程软件的主界面中独立出来(如下所示)，关闭独立使用的工具可以再次返回主界面。



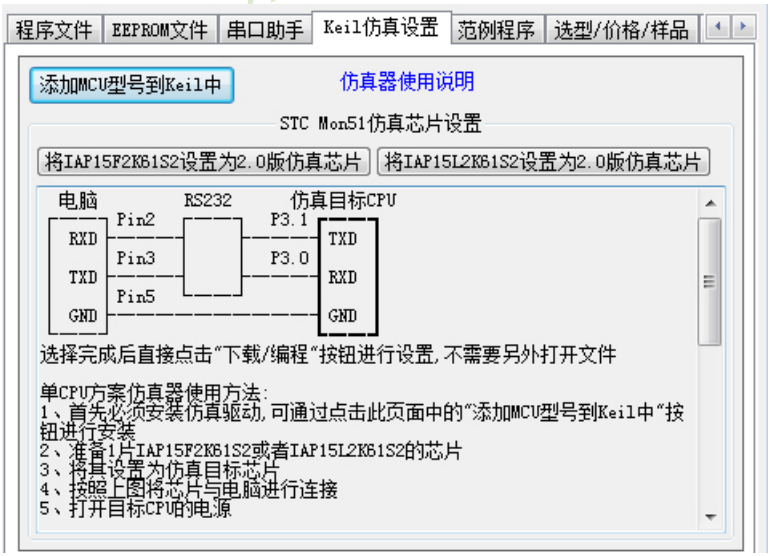
最新的STC-ISP-V6.65软件集成了波特率计算器，利用波特率计算器可以很方便地求出波特率，并可以生成相应的代码(C或ASM代码)。波特率计算器界面如下所示：

最新的STC-ISP-V6.65软件还集成了定时器计算器，定时器计算器也可以生成相应的代码(C或ASM代码)，根据用户的设置对定时器的各相关寄存器进行初始化。定时器计算器界面如下所示：

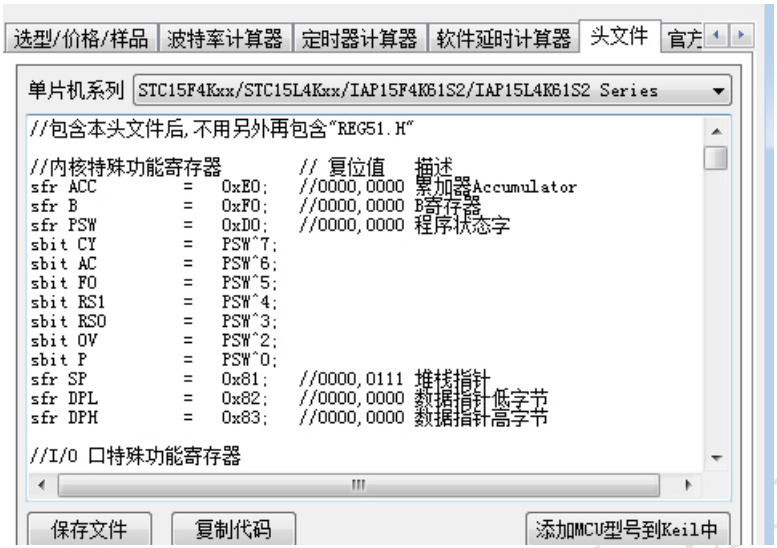
另外，最新的STC-ISP-V6.65软件还集成了软件延时计算器，软件延时计算器也可以生成相应的代码(C或ASM代码)，根据用户的设置可以生成相应的延时子函数。软件延时计算器界面如下所示：

除串口助手外，波特率计算器、定时器计算器、软件延时计算器都可以从STC-ISP下载编程软件的主界面中独立出来，关闭独立使用的工具可以再次返回主界面。

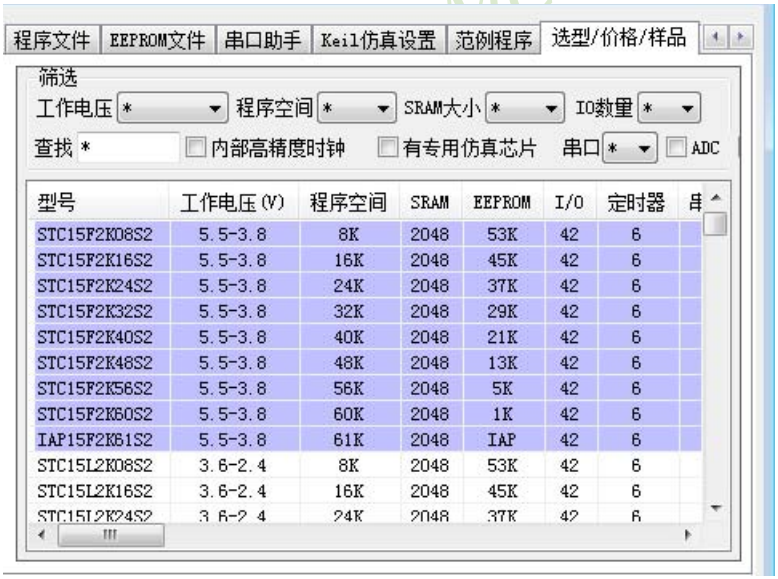
最新的STC-ISP-V6.65软件还设计了“Keil仿真设置”选项，如下图所示



最新的STC-ISP-V6.65软件还包含了头文件，供用户查询和复制。头文件如下所示：



另外，用户还可以在最新的STC-ISP-V6.65软件中查询STC系列单片机的选型和价格。



15.2.8 STC-ISP控制软件(Ver6.65)发布项目程序使用说明

发布项目程序功能主要是将用户的程序代码与相关的选项设置打包成为一个可以直接对目标芯片进行下载编程的超级简单的用户自己界面的可执行文件。

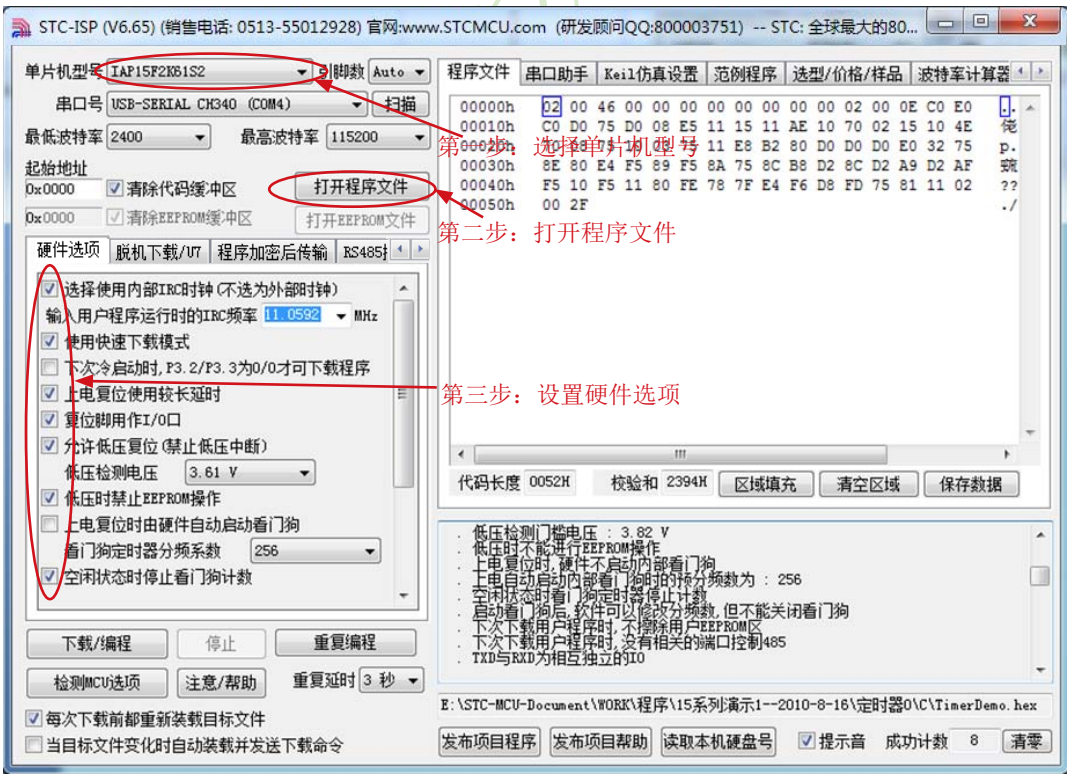
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的ID号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行(防止烧录人员将程序轻易从电脑盗走,如通过网络发走,如通过U盘烤走,防不胜防,当然盗走你的电脑那就没办法那,所以STC的脱机下载工具比电脑烧录安全,能限制可烧录芯片数量,让前台文员小姐烧,让老板娘烧都可以)，拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的ID号后，那么用户代码只能下载到具有相应ID号的目标芯片中(对于一台设备要卖几千万的产品特别有用---坦克,可以发给客户自己升级,不需冒着生命危险跑到战火纷飞的伊拉克升级软件啦)，对于ID号不一致的其它芯片，不能进行下载编程。

发布项目程序详细的操作步骤如下：

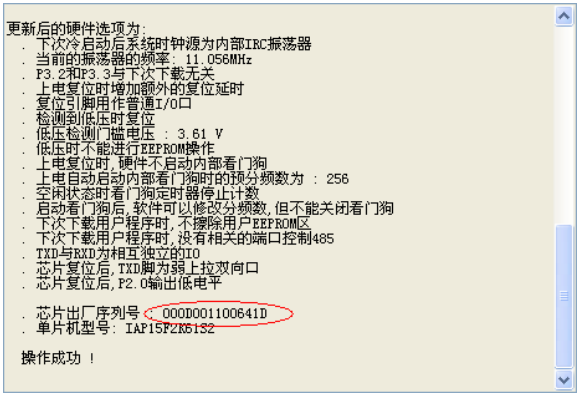
1、首先选择目标芯片的型号

2、打开程序代码文件

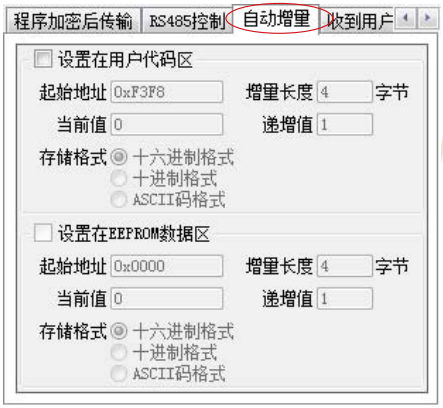
3、设置好相应的硬件选项



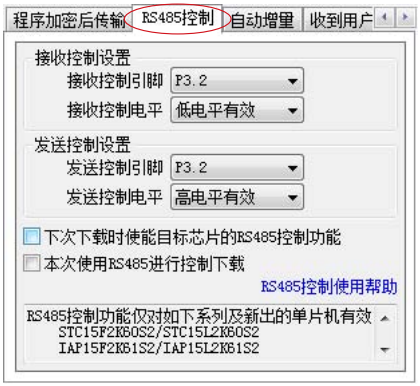
4、试烧一下芯片，并记下目标芯片的ID号，如下图所示，该芯片的ID号即为“000D001100641D”（如不需要对目标芯片的ID号进行校验，可跳过此步）



5、设置自动增量（如不需要自动增量，可跳过此步

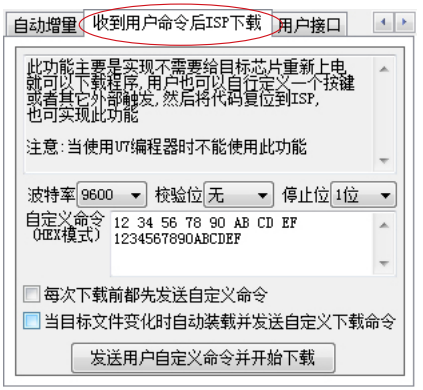


6、设置RS485控制信息（如不需要RS485控制，可跳过此步）

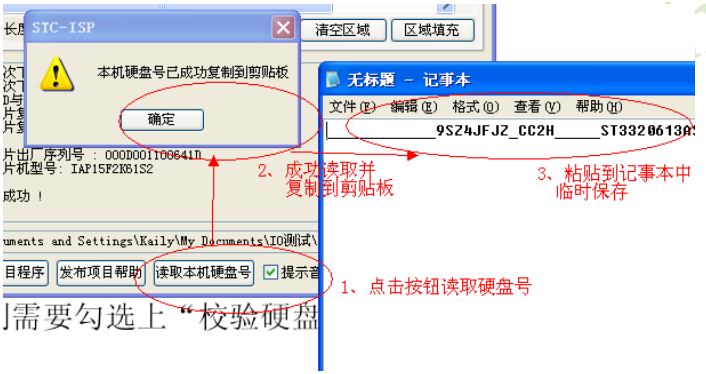


MCU Limited

7、设置“收到用户命令后ISP下载”（如不需要此功能，可跳过此步）



8、点击界面上的“读取本机硬盘号”按钮，并记下目标电脑的硬盘号（如不需要对目标电脑的硬盘号进行校验，可跳过此步）

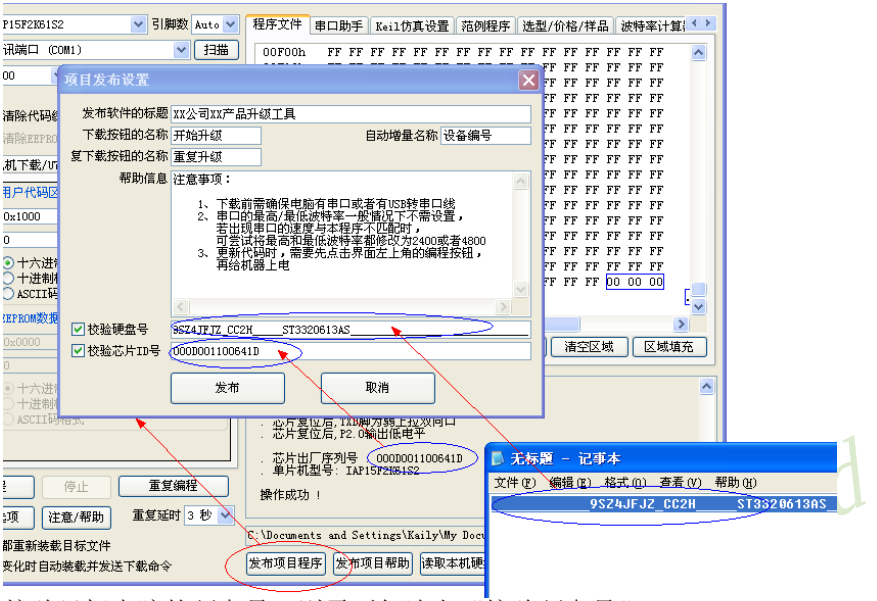


9、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

10、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息

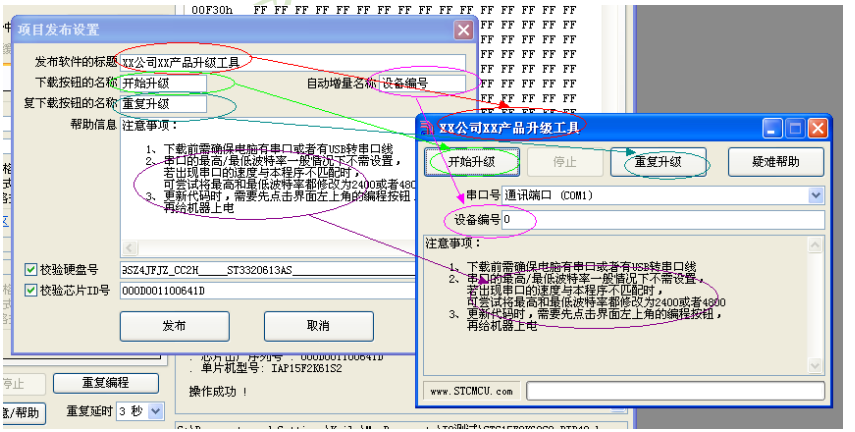
11、若需要校验目标电脑的硬盘号,则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记下的目标电脑的硬盘号

12、若需要校验目标芯片的ID号，则需要勾选上“校验芯片ID号”，并在后面的文本框内输入前面所记下的目标芯片的ID号



校验目标电脑的硬盘号，则需要勾选上“校验硬盘号”

13、最后点击发布按钮，将项目发布程序保存，即可得到相应的可执行文件。如下图，设置界面中所定制的内容与发布文件是一一对应的。



注意：

校验硬盘号与校验目标芯片ID号的功能仅对如下系列及新出的单片机有效：

STC15F2K60S2/STC15L2K60S2

IAP15F2K61S2/IAP15L2K61S2

STC15F101W/STC15L101W

IAP15F105W/STC15L105W

STC15W104SW/IAP15W105W

15.2.9 “程序加密后传输”功能说明
——防止烧录时通过串口分析出程序代码

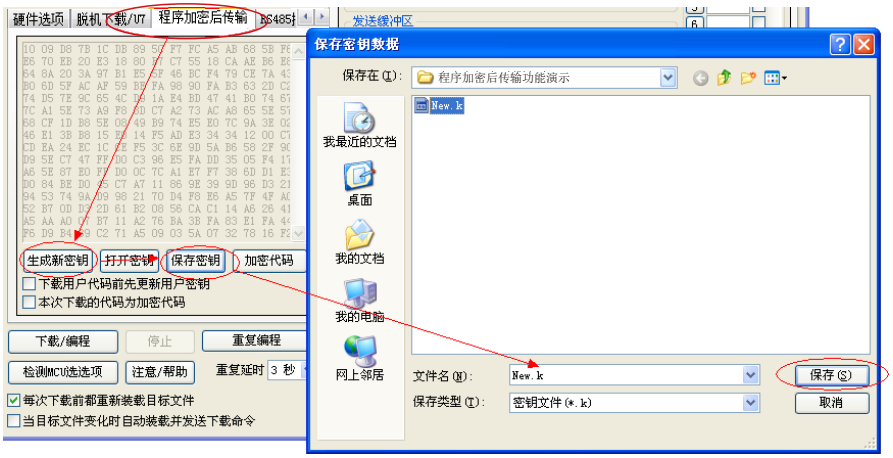
目前，所有的普通串口下载烧录编程都是采用明码通信的(电脑和目标芯片通信时,或脱机下载板和目标芯片通信时)，问题: 如果烧录人员通过分析下载烧录编程时串口通信的数据,高手是可以在烧录时在串口上引2根线出来,通过分析串口通信的数据分析出实际的用户程序代码的。当然用STC的脱机下载板烧程序总比用电脑烧程序强(防止烧录人员将程序轻易从电脑盗走,如通过网络发走,如通过U盘拷走,防不胜防,当然盗走你的电脑那就没办法那,所以STC的脱机下载工具比电脑烧录安全,让前台文员小姐烧,让老板娘烧都可以)。即使是STC全球首创的脱机下载工具,对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析串口通信的数据，分析出实际的用户程序代码，也是没有办法达到要求的，这就需要用到最新的STC15系列单片机所提供的“程序加密后传输”功能。目前，我司是全球第一家可以防范用户将程序代码给烧录人员烧录时烧录人员通过串口分析出目标程序代码的公司。

“程序加密后传输”功能是用户先将程序代码通过自己的一套专用密钥进行加密，然后将加密后的代码再通过串口下载，此时下载传输的是加密文件，通过串口分析出来的是加密后的乱码,如不通过派人潜入你公司盗窃你电脑里面的加密密钥,就无任何价值,便可起到防止在烧录程序时被烧录人员通过监测串口分析出代码的目的。

“程序加密后传输”功能的使用需要如下的几个步骤：

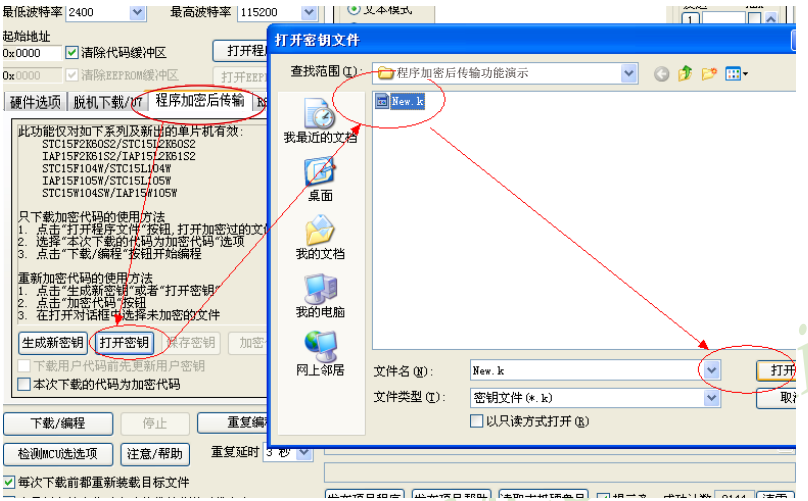
1、生成并保存新的密钥

如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的256字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的的密钥文件（注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得），例如我们将密钥保存为“New.k”。



2、对代码文件加密

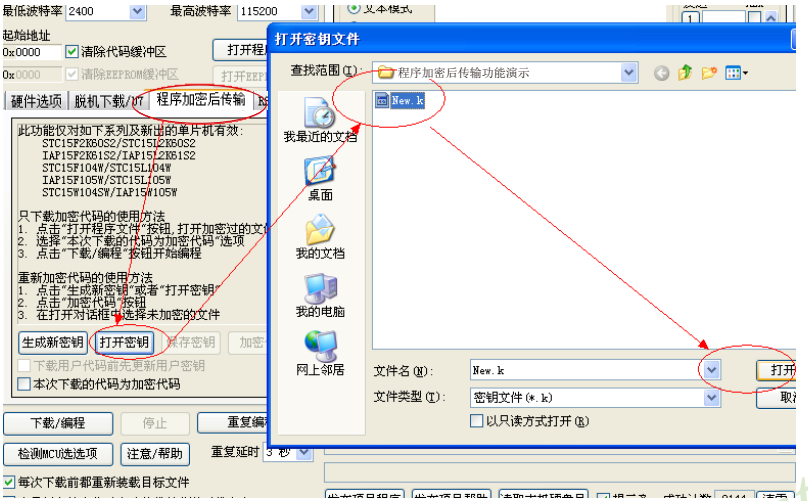
加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“自定义加密下载”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。



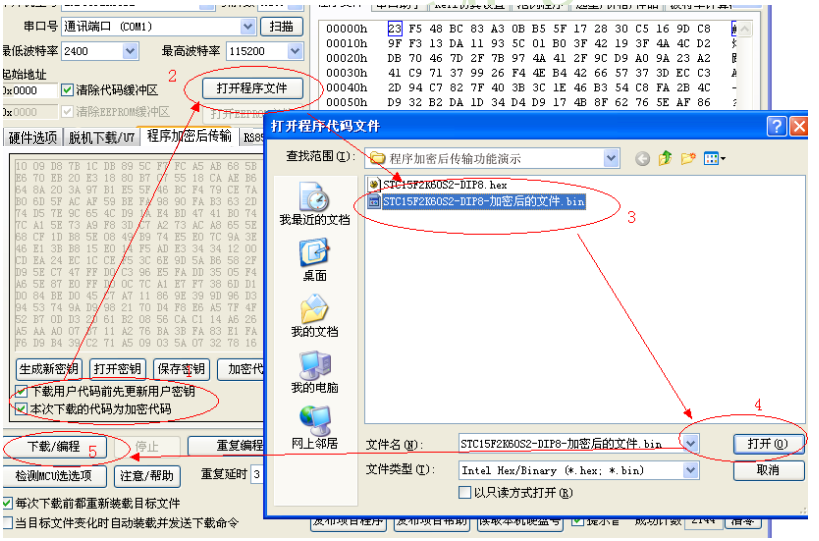
然后返回到“程序加密后传输”页面中点击“加密代码”按钮，如下图所示，首先会弹出“打开源文件（未加密）”的对话框，此时选择的是原始的未加密的代码文件。

3、将用户密钥更新到目标芯片中

更新密钥前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。

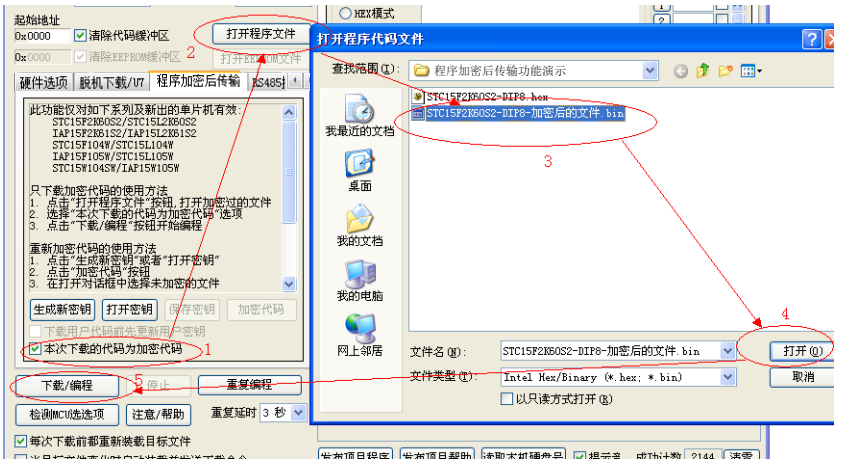


密钥打开后，如下图所示，勾选上“下载用户代码前先更新用户密钥”选项和“本次下载的代码为加密代码”的选项，然后打开我们之前加密过后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载完成即可更新用户密钥。



4、加密更新用户代码

密钥更新成功后，目标芯片便具有接收加密代码并还原的功能。此时若需要再次升级/更新代码，则只需要参考第二步的方法，将目标代码进行加密，然后如下图



首先在“程序加密后传输”页面中选择“本次下载的代码为加密代码”的选项（“下载用户代码前先更新用户密钥”选项不需要选了），然后打开我们之前加过密后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载即可完成用用户自己专用的加密文件更新用户代码的目的(防止在烧录程序时被烧录人员通过监测串口分析出代码的目的)。

注意：

“程序加密后传输”功能仅对如下系列及新出的单片机有效：

STC15F2K60S2/STC15L2K60S2

IAP15F2K61S2/IAP15L2K61S2

STC15F101W/STC15L101W

IAP15F105W/STC15L105W

STC15W104SW/IAP15W105W

STC MCU Limited

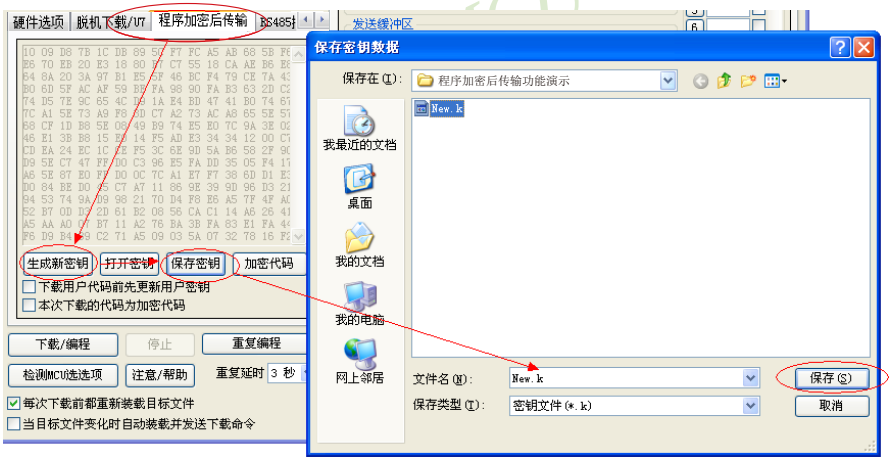
15.2.10 "发布项目程序"+"程序加密后传输"结合使用

“发布项目程序”与“程序加密后传输”两项新的特殊功能可以结合在一起使用。首先“程序加密后传输”可以确保用户代码在烧录编程时串口通信传输过程当中的保密性，而”发布项目程序”可实现让最终使用者远程升级功能（方案公司的人员不需要亲自到场）。所以两项功能结合起来使用，非常适用于方案公司/生产商在软件需要更新时,让最终使用者自己对终端产品进行软件更新的目的, 又确保现场烧录人员无法通过串口分析出有用程序, 强烈建议方案公司使用。

下面用具体的实例来举例说明”发布项目程序”与“程序加密后传输”结合使用的方法，首先讲解代码的加密以及加密芯片的制作方法

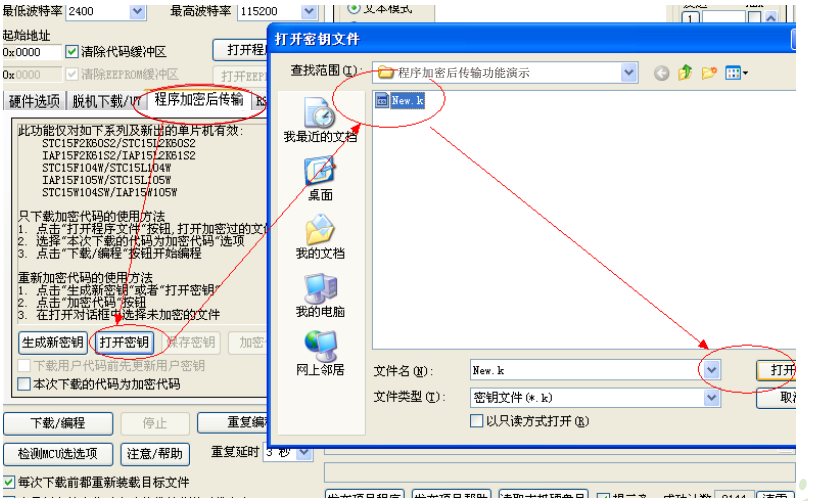
1、生成并保存新的密钥

如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的256字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的的密钥文件（注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得）。比如我们将密钥保存为“New.k”。

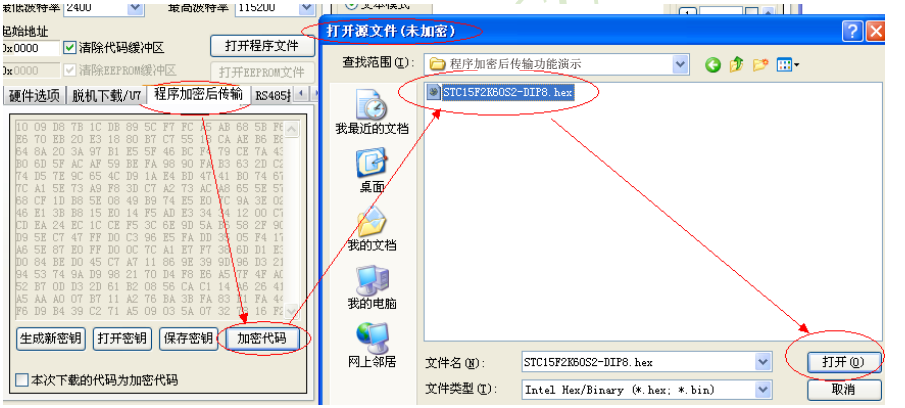


2、代码文件加密

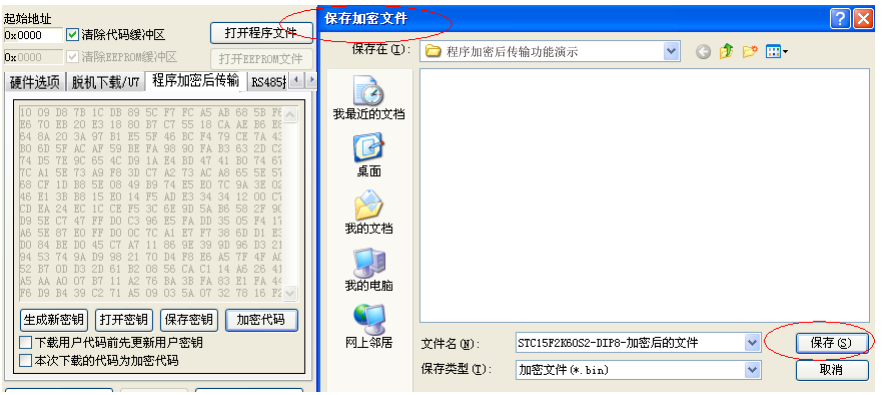
加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“自定义加密下载”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。



然后返回到“程序加密后传输载”页面中点击“加密代码”按钮，如下图所示，首先会弹出“打开源文件（未加密）”的对话框，此时选择的是原始的未加密的代码文件

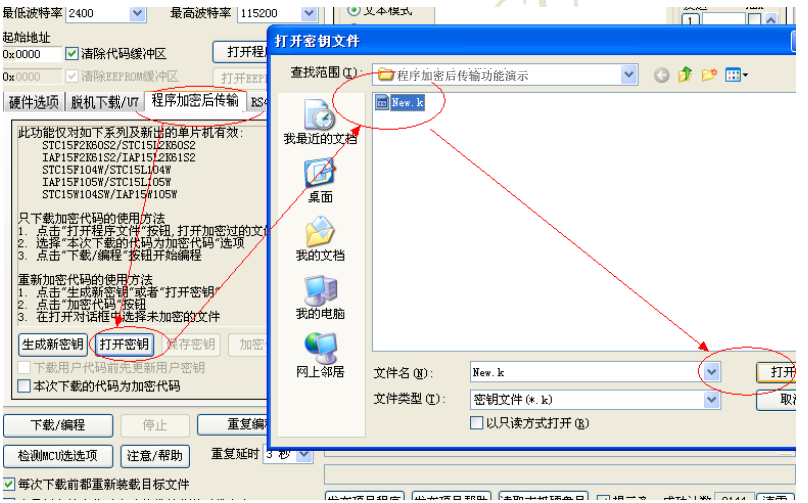


点击打开按钮后，马上有会弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。

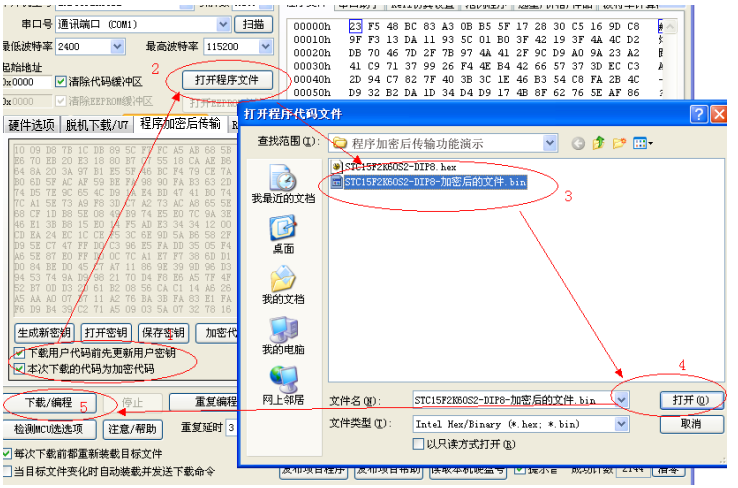


3、将用户密钥更新到目标芯片中

更新密钥前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。



密钥打开后，如下图所示，勾选上“下载用户代码前先更新用户密钥”选项和“本次下载的代码为加密代码”的选项，然后打开我们之前加密过后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载完成即可更新用户密钥。



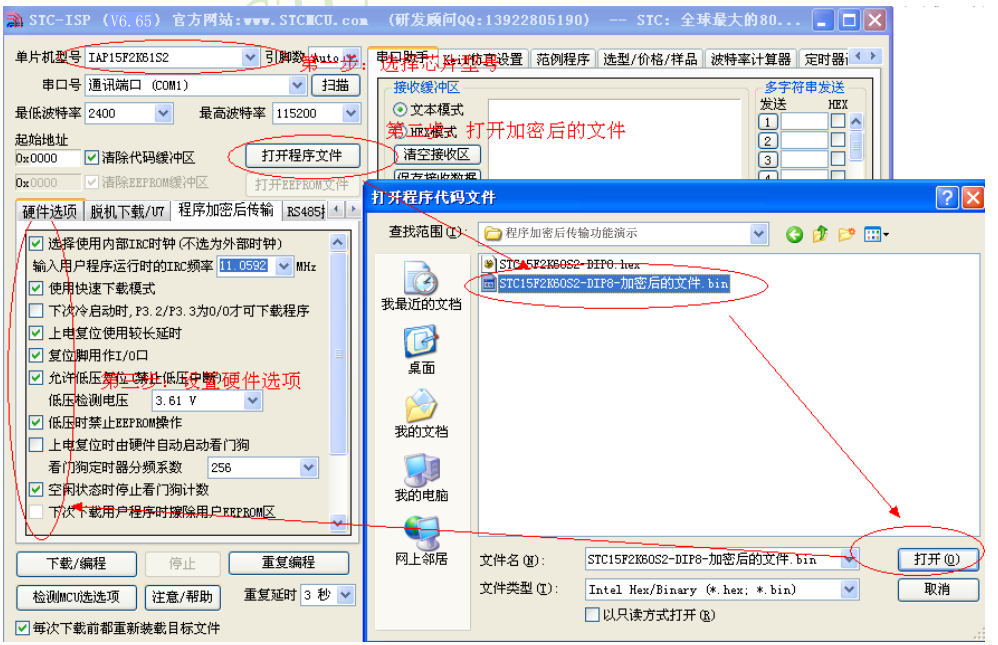
经过上面的三步，此时的目标芯片便具有还原加密代码的功能。便可将目标芯片提供给终端客户使用。

下面讲解如何发布加密项目程序

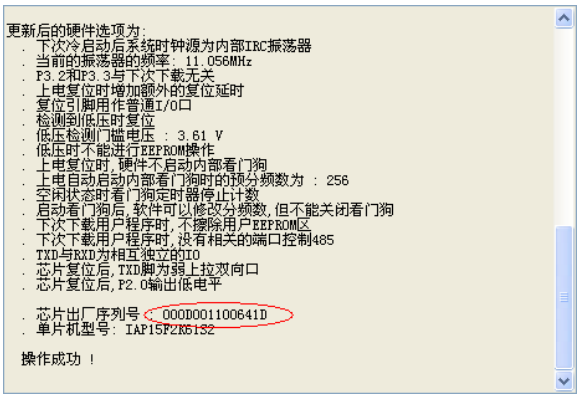
1、首先选择目标芯片的型号

2、打开程序代码文件

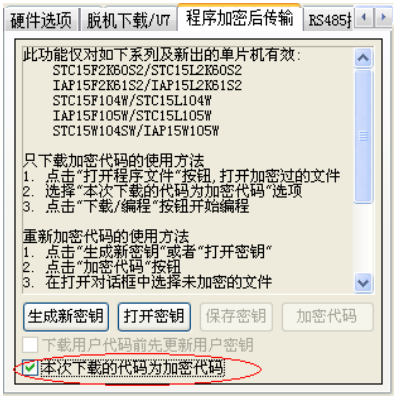
3、设置好相应的硬件选项



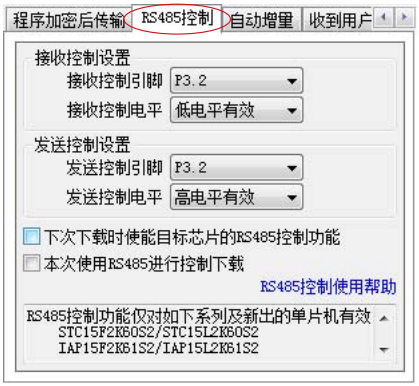
4、试烧一下芯片，并记下目标芯片的ID号，如下图所示，该芯片的ID号即为“000D001100641D”（如不需要对目标芯片的ID号进行校验，可跳过此步）



5、在“程序加密后传输”页面中选择“本次下载的代码为加密代码”选项（注意：加密下载时不支持自动增量）

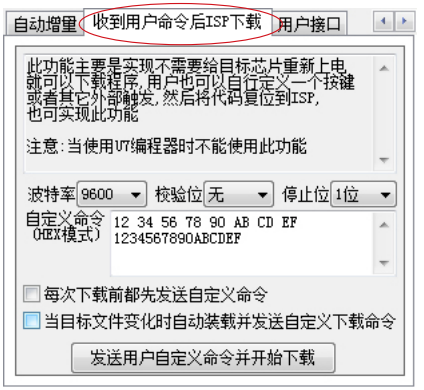


6、设置RS485控制信息（如不需要RS485控制，可跳过此步）

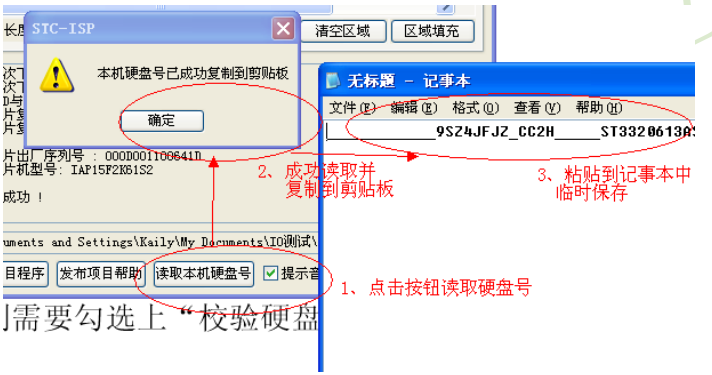


MCU Limited

7、设置“收到用户命令后ISP下载”（如不需要此功能，可跳过此步）



8、点击界面上的“读取本机硬盘号”按钮，并记下目标电脑的硬盘号（如不需要对目标电脑的硬盘号进行校验，可跳过此步）

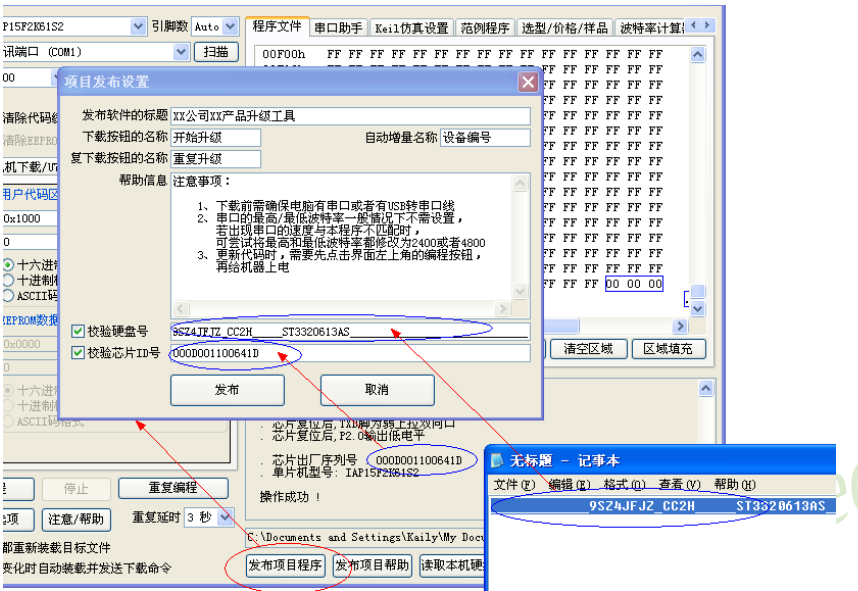


9、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

10、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息

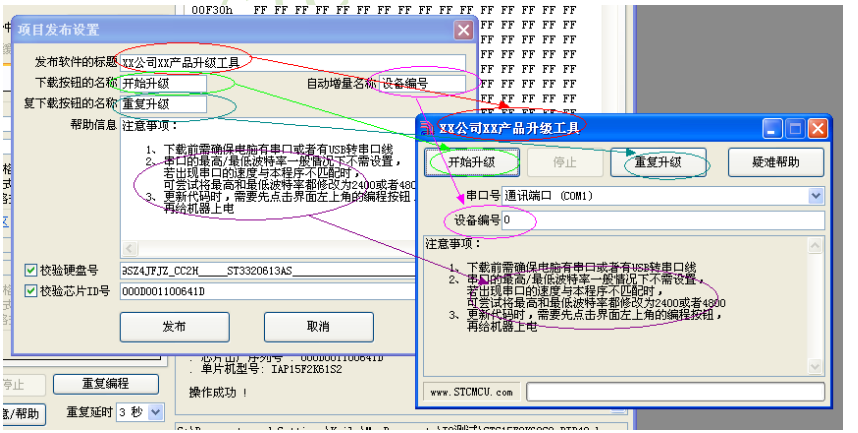
11、若需要校验目标电脑的硬盘号,则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记下的目标电脑的硬盘号

12、若需要校验目标芯片的ID号，则需要勾选上“校验芯片ID号”，并在后面的文本框内输入前面所记下的目标芯片的ID号



验目标电脑的硬盘号,则需要勾选上“校

13、最后点击发布按钮，将项目发布程序保存，即可得到相应的可执行文件。如下图，设置界面中所定制的内容与发布文件是一一对应的。



上面的整个步骤基本与发布项目程序的步骤相一致，唯一不同的地方是打开的不是原始文件，而是加密后的文件，而且一定要勾选上“本次下载的代码为加密代码”的选项。

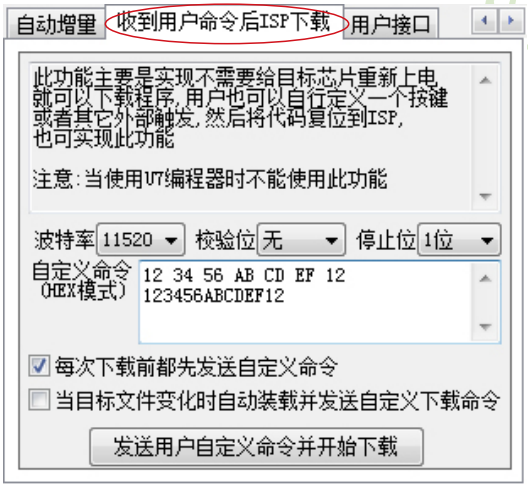
15.2.11 运行用户程序时收到用户命令后自动启动ISP下载(不停电)

“运行用户程序时收到用户命令后自动启动ISP下载”（即软件中的“收到用户命令后ISP下载”）与“程序加密后传输”是两种完全不同功能。相对“程序加密后传输”的功能而言，“运行用户程序时收到用户命令后自动启动ISP下载”的功能要简单一些。

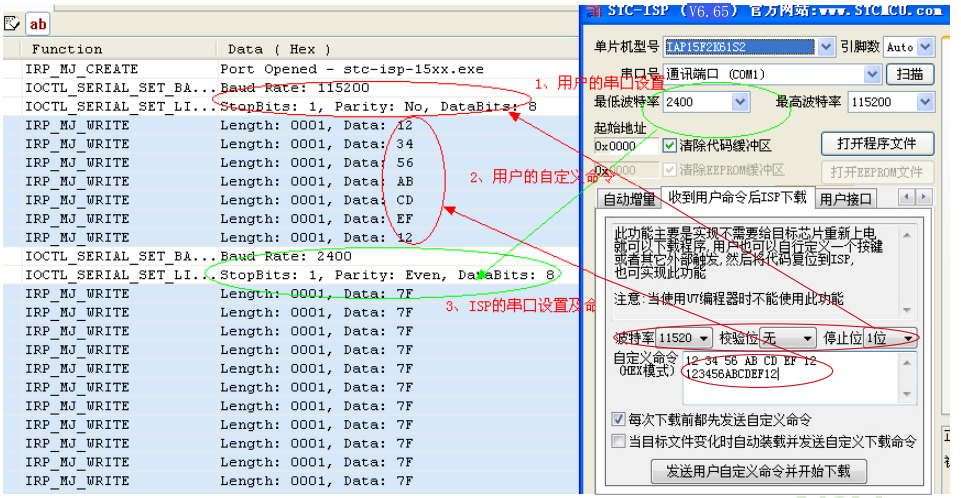
具体的功能为：电脑或脱机下载板在开始发送真正的ISP下载编程握手命令前，先发送用户自定义的一串命令（关于这一串串口命令，用户可以根据自己在应用程序中的串口设置来设置波特率、校验位以及停止位），然后再立即发送ISP下载编程握手命令。

“运行用户程序时收到用户命令后自动启动ISP下载”这一功能主要是在项目的早期开发阶段，实现不断电（不用给目标芯片重新上电）即可下载用户代码。具体的实现方法是：用户需要在自己的程序中加入一段检测自定义命令的代码，当检测到后，执行一句“MOV IAP_CONTR,#60H”的汇编代码或者“IAP_CONTR = 0x60;”的C语言代码，MCU就会自动复位到ISP区域执行ISP代码。

如下图所示，将自定义命令设置为波特率为115200、无校验位、一位停止位的命令序列：0x12、0x34、0x56、0xAB、0xCD、0xEF、0x12、。当勾选上“每次下载前都先发送自定义命令”的选项后，即可实现自定义下载功能



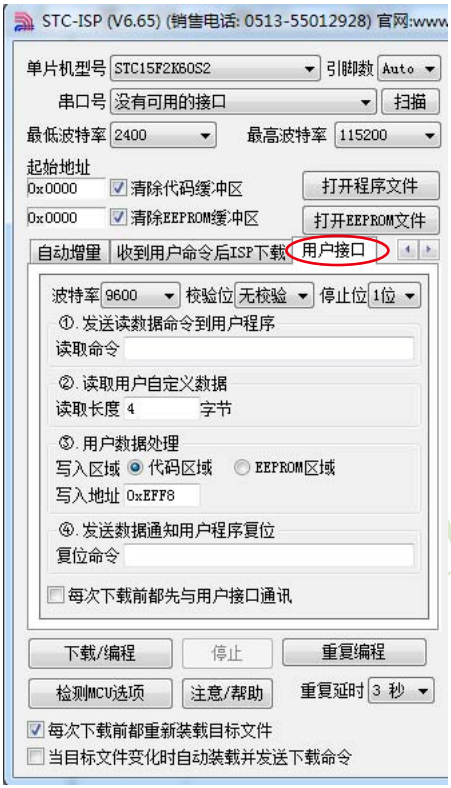
点击“发送用户自定义命令开始下载”或者点击界面左下角的“下载/编程”按钮，应用程序便会发送如下所示的串口数据



STC MCU Limited

15.2.12 用户接口

STC-ISP-V6.65下载编程软件新增了用户接口软件，如下图所示。用户接口功能主要实现了保留用户芯片中的重要信息（如：序列号）不被破坏的作用。



使用用户接口功能时，PC机或者U7编程器首先与用户单片机通讯，将单片机的重要信息(如：序列号等)读取出来并保存，然后用户可以设置发送给用户代码的自定义命令(如设置发送给用户代码的读数据命令或复位命令)，用户代码可以接收自定义命令。当用户代码收到复位命令后可以控制目标单片机自动复位，若用户未设置复位命令，则用户需手动给目标单片机重新上电，当目标单片机上电复位后，就开始更新代码了，此时更新的代码包括上述PC机或U7编程器所保存的重要信息和用户新代码，即将PC机或U7编程器所保存的重要信息和用户新代码一并写入了目标单片机中，从而实现了保留目标单片机中的重要信息不被破坏的目的。

注意：只有使用普通串口或USB转串口直接对单片机进行在线下载或者使用U7编程器进行脱机下载时，用户接口功能才可用；当使用U7编程器在线联机下载时，用户接口功能并不可用。

MCU Limited

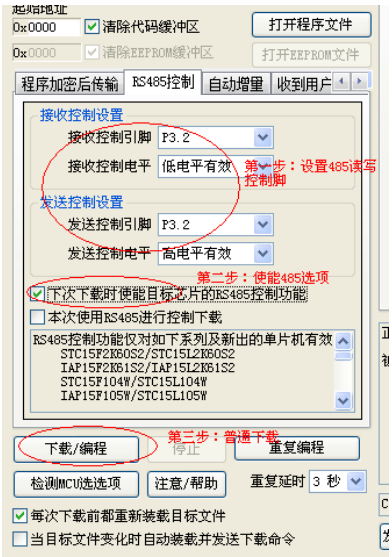
15.2.13 RS485控制使用说明

由于RS485相比RS232具有抑制共模干扰、传输距离长等优点，所以许多大型的工业设备都采用RS485进行串口通讯。但由于RS485采用的是差分信号，所以在进行串口通讯时，只能采用半双工的工作方式，必须使用1个或2个I/O口来控制RS485的发送和接收状态。当需要采用RS485来对STC的新版IC（支持RS485下载的单片机系列在后面会详细列出）进行ISP下载时，必须进行一些设置才可下载代码。

具体的操作步骤如下：

1、首先需要设置好相应的RS485控制端口，并勾选上“下次下载时使能目标芯片的RS485控制功能”这个选项

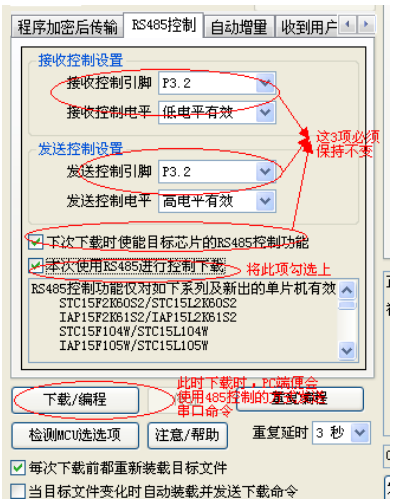
2、然后使用普通下载方式将RS485相关的硬件选项写入到目标芯片



3、经过前面两步的设置和编程，此时的目标芯片便具有了对RS485的控制功能。接下来仍需要保持RS485的控制选项不变，并勾选上"本次使用RS485进行控制下载"的选项（此选项的作用是使PC端也采用RS485的控制方式进行发送/接收串口数据）

4、再点击下载编程按钮，并对目标芯片重新上电即可实现使用RS485进行通信下载的功能

MCU Limited



RS485控制功能仅对如下系列及新出的单片机有效：
STC15F2K60S2/STC15L2K60S2
IAP15F2K61S2/IAP15L2K61S2
STC15F101W/STC15L101W
IAP15F105W/STC15L105W
STC15W104SW/IAP15W105W

特别注意：
若需要RS485控制功能，则每次都需要将RS485相关的配置设置正确，并勾选上“下次下载时使能目标芯片的RS485控制功能”这个选项，否则在下一次下载时将不具有RS485控制功能了

STC MCU Limited

15.3 若无仿真器，如何调试/开发用户程序

STC单片机部分系列无仿真器，如STC89xx系列、STC90xx系列等，但长沙菊阳微电子有限公司以及南京伟福实业有限公司均有通用的STC89xx、STC90xx系列单片机仿真器购买。当然部分STC单片机也有自己的仿真器，如最新的STC15系列。

现介绍在没有仿真器的情况下如何调试和开发用户程序，具体操作步骤如下：

1. 首先参照本手册当中的“用定时器1做波特率发生器”，调通串口程序，这样，要观察变量就可以自己写一小段测试程序将变量通过串口输出到电脑端的STC-ISP.EXE的“串口调试助手”来显示,也很方便。
2. 调通按键扫描程序(到处都有大量的参考程序)
3. 调通用户系统的显示电路程序，此时变量/寄存器也可以通过用户系统的显示电路显示了
4. 调通A/D检测电路(我们用户手册里面有完整的参考程序)
5. 调通PWM等电路(我们用户手册里面有完整的参考程序)

这样分步骤模块化调试用户程序，有些系统，熟练的8051用户，三天就可以调通了，难度不大的系统，一般一到二周就可以调通。

用户的串口输出显示程序可以在输出变量/寄存器的值之后，继续全速运行用户程序，也可以等待串口送来的“继续运行命令”，方可继续运行用户程序，这就相当于断点。这种断点每设置一个地方，就必须调用一次该显示寄存器/变量的程序，有点麻烦，但却很实用。

STC MCU Limited

15.4 STC仿真器说明指南(建议串口放在P3.6/P3.7或P1.6/P1.7上)

1. 仿真器的参考硬件图

（1）利用RS-232转换器连接电脑的的仿真应用线路图

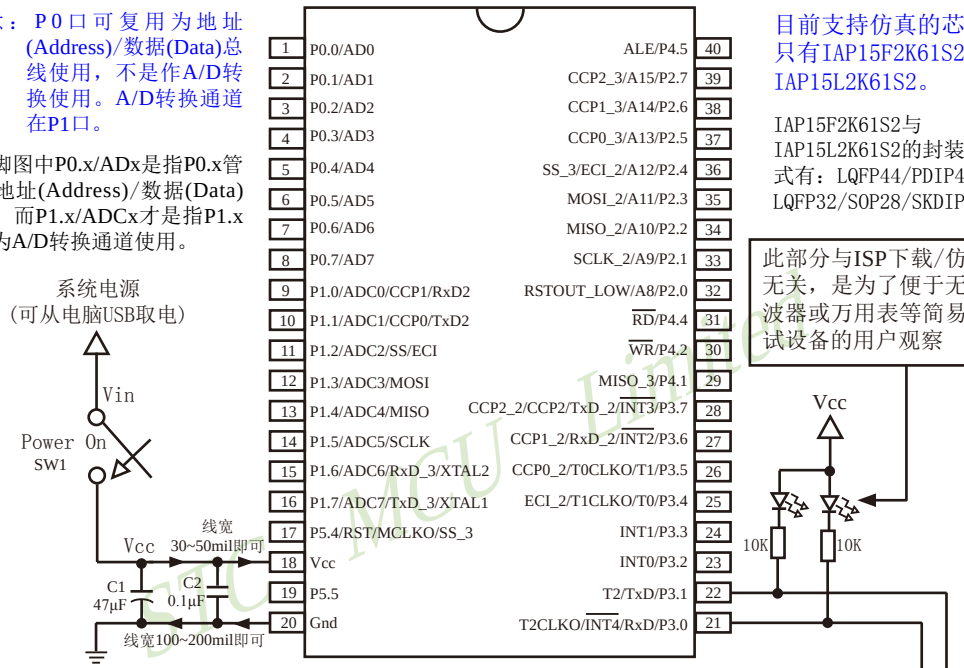
特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。

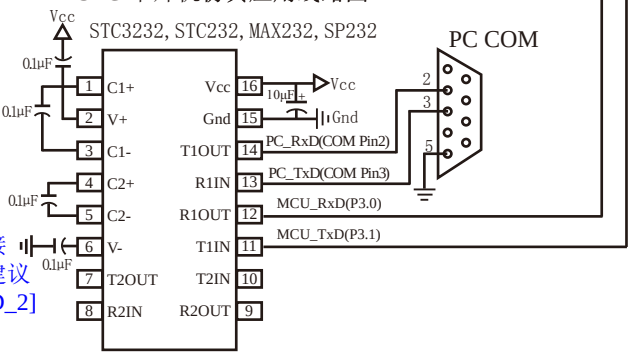
目前支持仿真的芯片只有IAP15F2K61S2与IAP15L2K61S2。

IAP15F2K61S2与IAP15L2K61S2的封装形式有：LQFP44/PDIP40/LQFP32/SOP28/SKDIP28

此部分与ISP下载/仿真无关，是为了便于无示波器或万用表等简易测试设备的用户观察



开始仿真前，须先给目标芯片上电，再点击Keil菜单栏中的Debug->Start/Stop Debug或按“Ctrl+F5”开始调试



STC 单片机仿真应用线路图

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口1放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路，当然也可以使用外部复位电路
P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振，当然也可以使用外部晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

（2）利用USB转串口连接电脑的仿真典型应用线路图

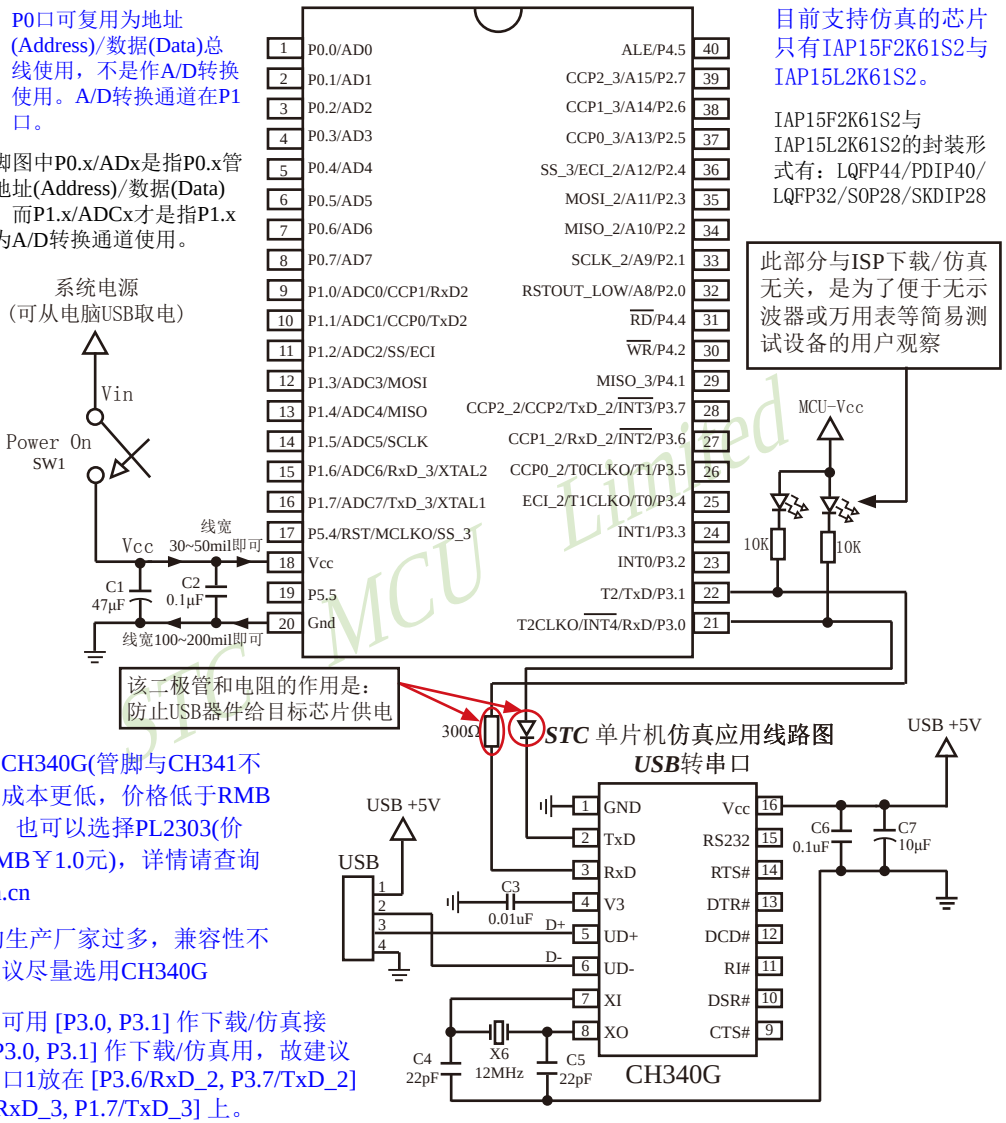
特别注意：P0口可复用为地址(Address)/数据(Data)总线使用，不是作A/D转换使用。A/D转换通道在P1口。

因此：管脚图中P0.x/ADx是指P0.x管脚可作为地址(Address)/数据(Data)总线使用，而P1.x/ADCx才是指P1.x管脚可作为A/D转换通道使用。

目前支持仿真的芯片只有IAP15F2K61S2与IAP15L2K61S2。

IAP15F2K61S2与IAP15L2K61S2的封装形式有：LQFP44/PDIP40/LQFP32/SOP28/SKDIP28

此部分与ISP下载/仿真无关，是为了便于无示波器或万用表等简易测试设备的用户观察



建议选用CH340G(管脚与CH341不兼容，但成本更低，价格低于RMB￥1.3元)，也可以选择PL2303(价格低于RMB￥1.0元)，详情请查询www.wch.cn

PL2303的生产厂家过多，兼容性不一致，建议尽量选用CH340G

注意：仅可用 [P3.0, P3.1] 作下载/仿真接口，因 [P3.0, P3.1] 作下载/仿真用，故建议用户将串口1放在 [P3.6/RxD_2, P3.7/TxD_2] 或 [P1.6/RxD_3, P1.7/TxD_3] 上。

内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

2. 软件环境

对于汇编语言程序,复位入口的程序必须是长跳转指令,可使用如下语句

```
        ORG     0000H                   ;复位入口地址
        LJMP    RESET                   ;使用LJMP指令
        …                               ;其它中断向量
        ORG     100H                    ;用户代码地址
RESET:                                  ;复位入口
        …                               ;用户代码
```

3. 仿真代码占用的资源

程序空间 :仿真代码占用程序区最后6K字节的空间

如果用IAP15F2K61S2/IAP15L2K61S2单片机仿真时，用户程序只能占55K（0x0000~0xDBFF）空间, 用户程序不要使用从0xDC00到0xF3FF的6K字节空间

常规RAM（data,idata）：0字节

XRAM(xdata) : 768字节(0x0400 – 0x06FF, 用户在程序中不要使用)

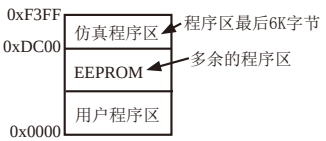
I/O : P3.0 / P3.1

用户在程序中不得操作P3.0/P3.1, 不要使用INT4/T2CLKO/P3.0, 不要使用T2/P3.1

不要使用外部中断INT4,不要使用T2的时钟输出功能，不要使用T2的外部计数功能

对于IAP型号单片机，对EEPROM的操作是通过对多余不用的程序区进行IAP模拟实现的，此部分要修改程序(IAP起始地址)。

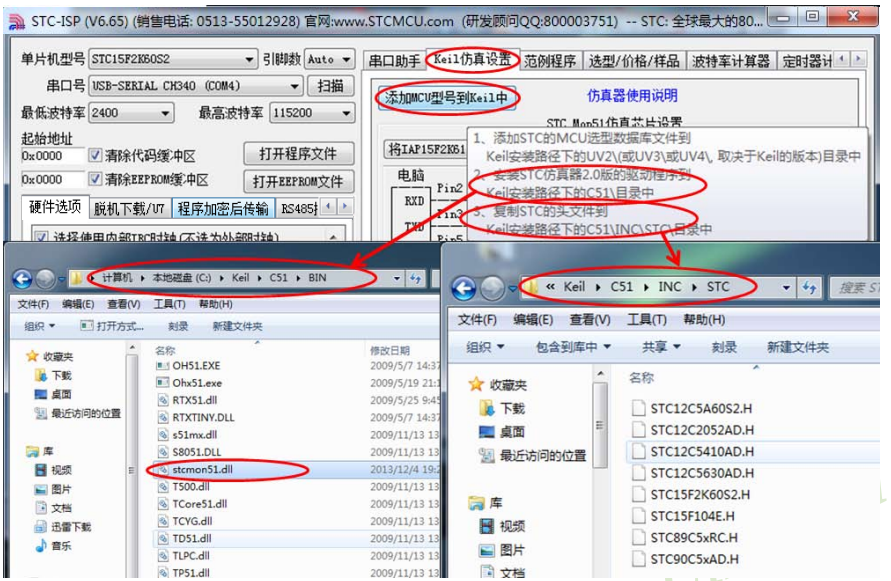
如IAP15F2K61S2单片机的EEPROM区的位置如右图所示。



4. 仿真器操作步骤

（1）安装Keil版本的仿真驱动，如下图所示:

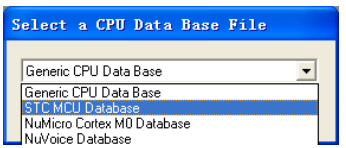
STC MCU Limited



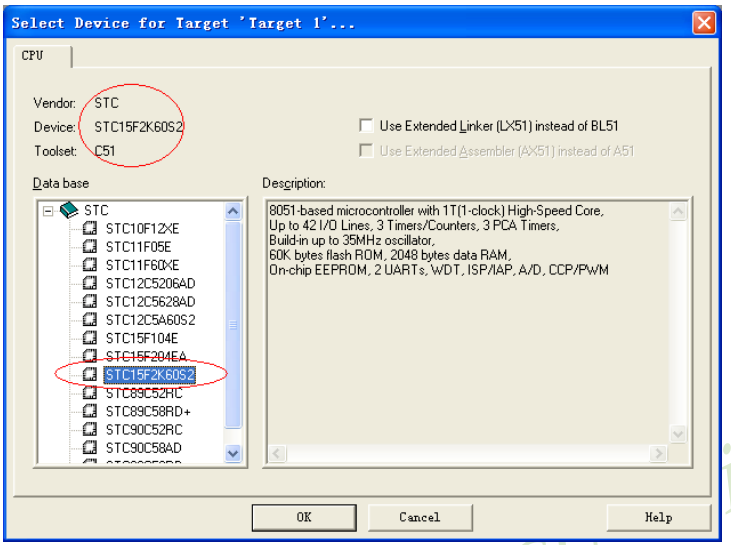
如上图，首先选择“Keil仿真设置”页面，点击“添加MCU型号到Keil中”，在出现的如下的目录选择窗口中，定位到Keil的安装目录(一般可能为“C:\Keil\”)，“确定”后出现下图中右边所示的提示信息，表示安装成功。添加头文件的同时也会安装STC的Monitor51仿真驱动STCMON51.DLL，驱动与头文件的的安装目录如上图所示。

（2）在Keil中创建项目

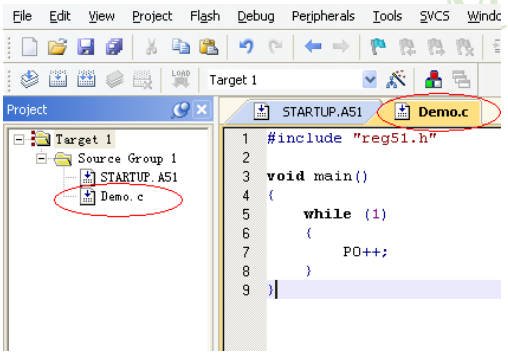
若第一步的驱动安装成功，则在Keil中新建项目时选择芯片型号时，便会有“STC MCU Database”的选择项，如下图



然后从列表中选择响应的MCU型号（目前STC支持仿真的型号只有STC15F2K60S2），所以我们在此选择“STC15F2K60S2”的型号，点击“确定”完成选择



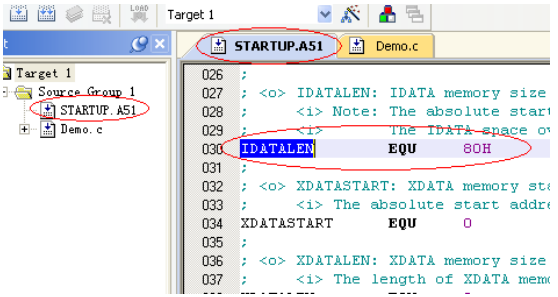
添加源代码文件到项目中，如下图：



保存项目，若编译无误，则可以进行下面的项目设置了

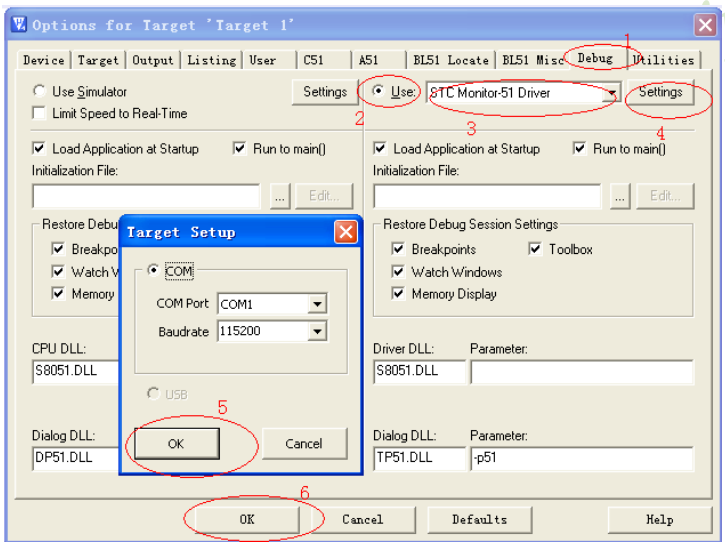
附加说明一点：

当创建的是C语言项目，且有将启动文件“STARTUP.A51”添加到项目中时，里面有一个命名为“IDATALEN”的宏定义，它是用来定义IDATA大小的一个宏，默认值是128，即十六进制的80H，同时它也是启动文件中需要初始化为0的IDATA的大小。所以当IDATA定义为80H，那么STARTUP.A51里面的代码则会将IDATA的00-7F的RAM初始化为0；同样若将IDATA定义为0FFH，则会将IDATA的00-FF的RAM初始化为0。



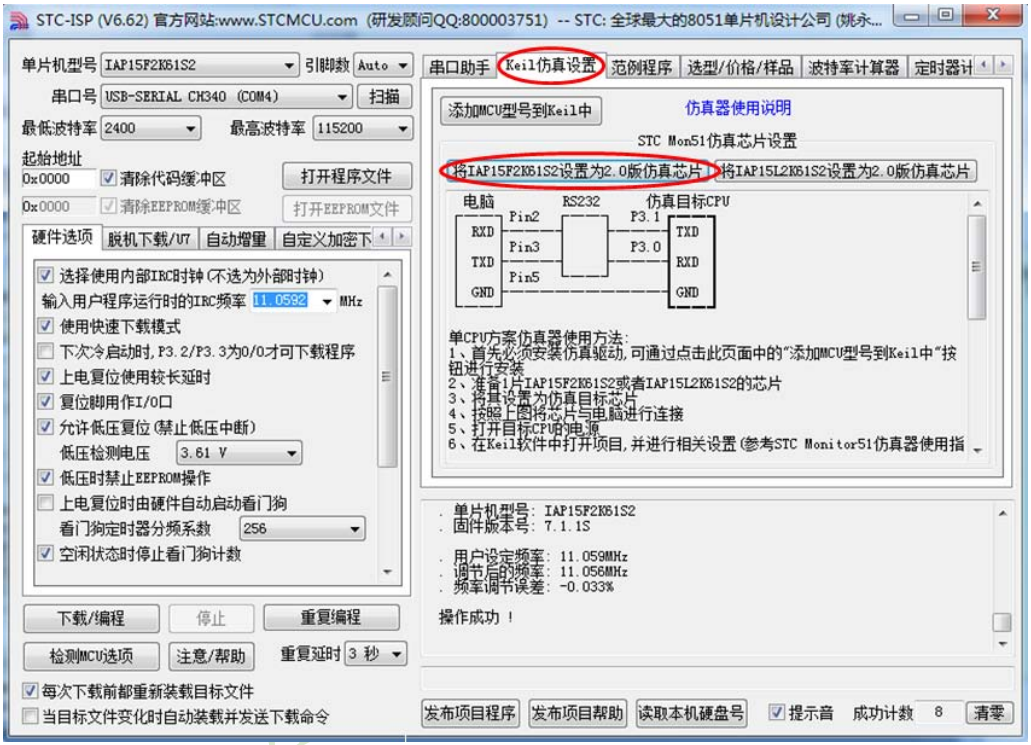
虽然STC15F2K60S2系列的单片机的IDATA大小为256字节（00-7F的DATA和80H-FFH的IDATA），但由于STC15F2K60S2在RAM的最后17个字节有写入ID号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将IDATALEN定义为256。

（3）项目设置，选择STC仿真驱动



如上图，首先进入到项目的设置页面，选择“Debug”设置页，第2步选择右侧的硬件仿真“Use …”，第3步，在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项，然后点击“Settings”按钮，进入下面的设置画面，对串口的端口号和波特率进行设置，波特率一般选择115200或者57600。到此设置便完成了。

（4）创建仿真芯片



准备一颗IAP15F2K61S2或者IAP15L2K61S2的芯片，并通过下载板连接到电脑的串口，然后如上图，选择正确的芯片型号，然后进入到“Keil仿真设置”页面，点击“将IAP15F2K61S2设置为2.0版仿真芯片”按钮或者“将IAP15L2K61S2设置为2.0版仿真芯片”按钮，当程序下载完成后仿真器便制作完成了。

（5）开始仿真

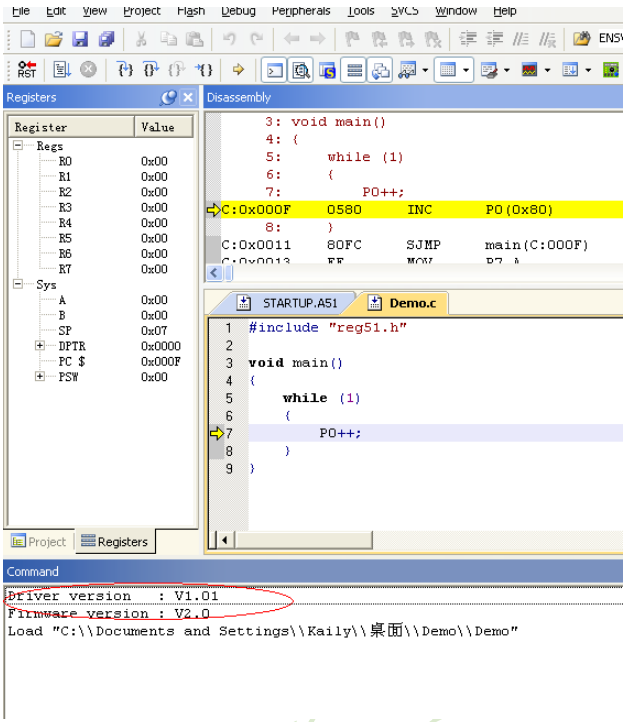
将制作完成的仿真芯片通过串口与电脑相连接，并给目标芯片上电。

将前面我们所创建的项目编译至没有错误后，按“Ctrl+F5”开始调试。

若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号

断点设置的个数目前最大允许20个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。

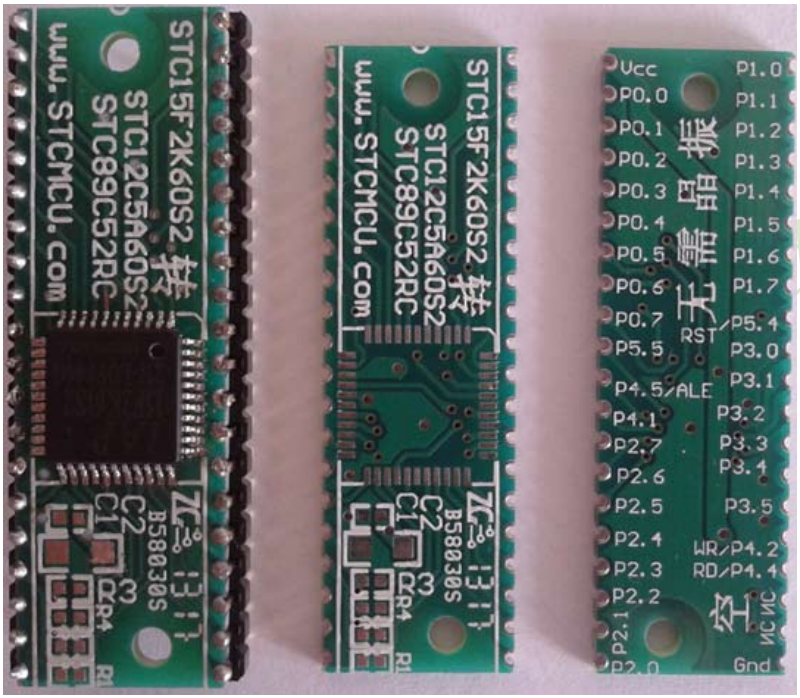
注意：开始仿真前，须先给目标芯片上电，再点击Keil菜单栏中的Debug->Start/Stop Debug或按“Ctrl+F5”开始调试


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Regs
R0 0x00
R1 0x00
R2 0x00
R3 0x00
R4 0x00
R5 0x00
R6 0x00
R7 0x00
Sys
A 0x00
B 0x00
SP 0x07
DPTR 0x0000
PC $ 0x000F
PSW 0x00
Project Registers
Disassembly
3: void main()
4: {
5: while (1)
6: {
7: P0++;
C:0x000F 0580 INC P0(0x80)
8: }
C:0x0011 80FC SJMP main(C:000F)
STARTUP.A51 Demo.c
1 #include "reg51.h"
2
3 void main()
4 {
5 while (1)
6 {
7 P0++;
8 }
9 }
Command
Driver version : V1.01
Firmware version : V2.0
Load "C:\\Documents and Settings\\Kaily\\桌面\\Demo\\Demo"

Limited
STC

15.5 如何让传统的8051单片机学习板可仿真

传统的8051单片机学习板不具有仿真功能，让传统的8051单片机学习板可仿真需要借助转换板，转换板的实物图如下图所示，转换后的引脚排布与传统8051的脚位基本一致，从而可以实现标准8051学习板的仿真功能。



完整转换板　　完整转正面　　完整转反面

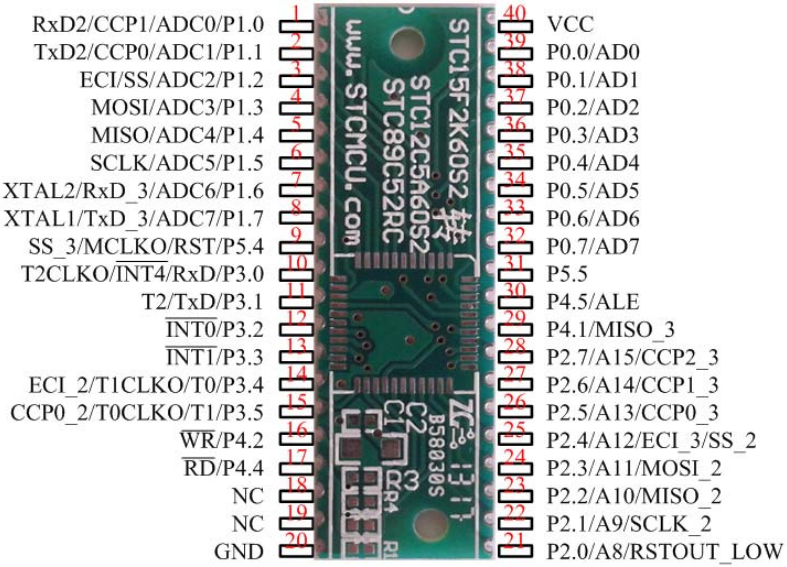
该转换板可进行IAP15F2K61S2/STC15F2K60S2转STC89C52RC/STC89C58RD+系列仿真用、IAP15F2K61S2/STC15F2K60S2转STC90C52RC/STC90C58RD+系列仿真用、IAP15F2K61S2/STC15F2K60S2转STC10F08XE/STC11F60XE系列仿真用、及IAP15F2K61S2/STC15F2K60S2转STC12C5A60S2系列仿真用。

目前，我公司只是小批量生产此转换板，供客户快速验证用，如需要我们提供样板，售价为：空板：1元人民币；

转换板+主控芯片（IAP15F2K61S2/STC15F2K60S2）：6元人民币。

若用户自己批量生产此板，成本价可控制在0.40元以下。新产品开发请直接使用STC15F2K60S2/IAP15F2K61S2系列来开发

下图为转换板功能示意图（IAP15F2K61S2转STC89C52/90C52/12C5A60S2仿真用转换板）



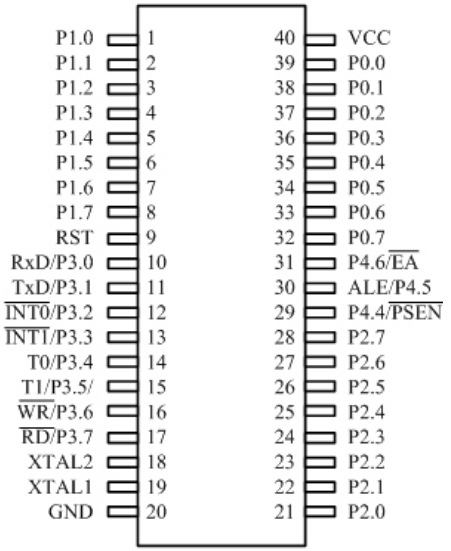
注意：

由于内置高精准R/C时钟(5MHz ~ 40MHz可设)，因此不需要外部晶振；

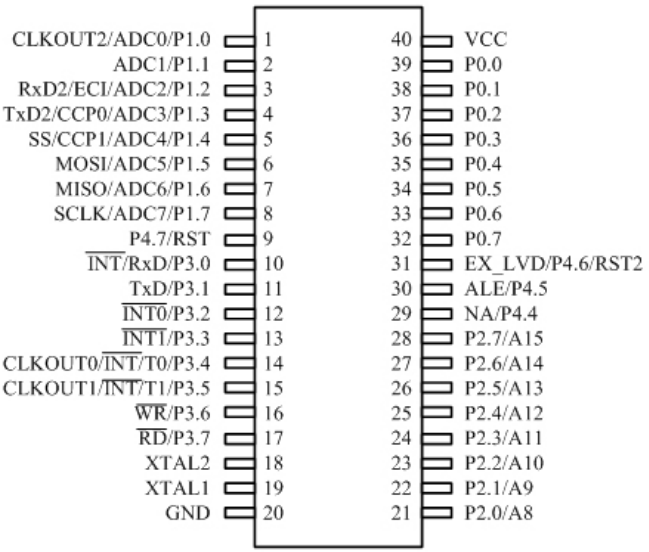
XTAL1和XTAL2是空的

WR和RD 是(WR/P4.2和RD/P4.4)，而不是传统的(WR/P3.6和RD/P3.7)

下面为STC89C52RC和STC12C5A60S2的脚位分布图



STC89C52RC-PDIP40



STC12C5A60S2-PDIP40

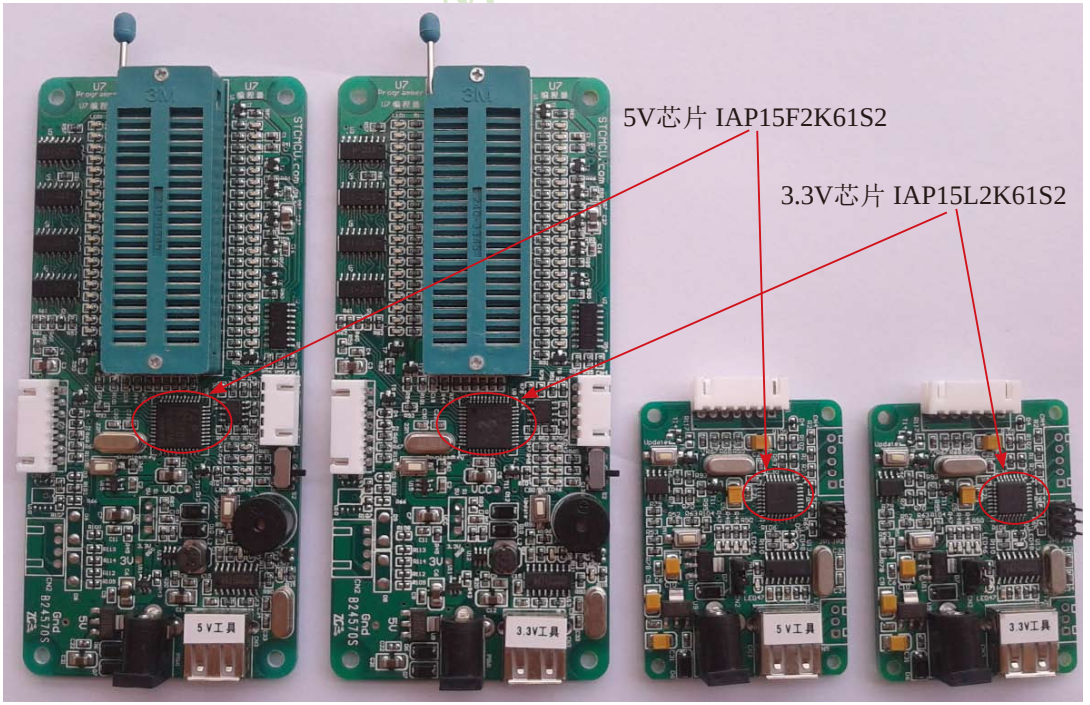
15.6 USB型联机/脱机下载工具U7/U7-S1

U7/U7-S1是一款集在线联机下载和脱机下载于一体的编程工具系列。应用范围可支持STC目前的全部系列的MCU, Flash程序空间和EEPROM数据空间不受限制。支持包括如下和即将推出的STC全系列芯片：

STC15F2K60S2/STC15L2K60S2系列
STC15F408AD/STC15L408AD系列
STC15W201S系列
STC15F104W/STC15L104W系列
STC15F104E/STC15L104E
STC15F204EA/STC15L204EA
STC10Fxx/STC10Lxx系列
STC11Fxx/STC11Lxx系列
STC12C5Axx/STC12LE5Axx系列
STC12C52xx/STC12LE52xx系列
STC12C56xx/STC12LE56xx系列
STC12C54xx/STC12LE54xx系列
STC12Cx052/ STC12Cx052AD/STC12LEx052/STC12LEx052AD系列STC90xx/STC89xx系列

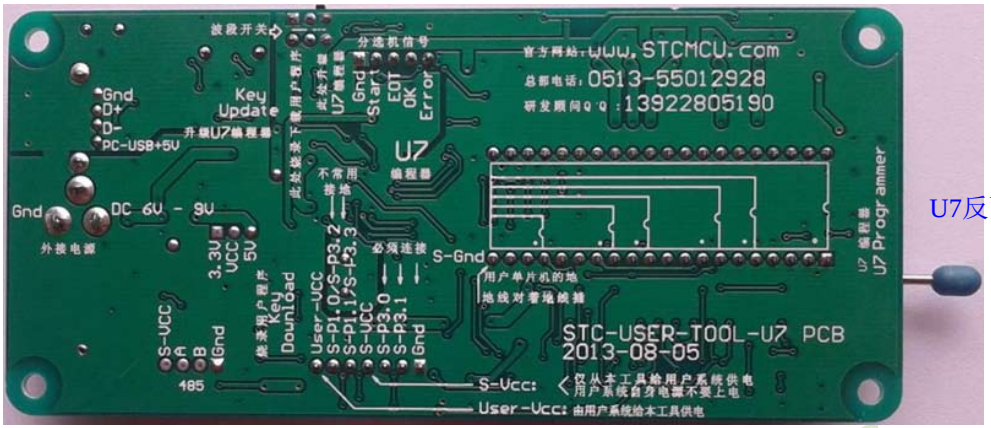
脱机下载工具可以在脱离电脑的情况下进行下载工作，可用于批量生产和远程升级。脱机下载板可支持自动增量、下载次数限制以及用户自定义加密下载等多种功能。

目前的U7/U7-S1系列工具均分为5V工具和3.3V工具两种，正面图如下图所示：

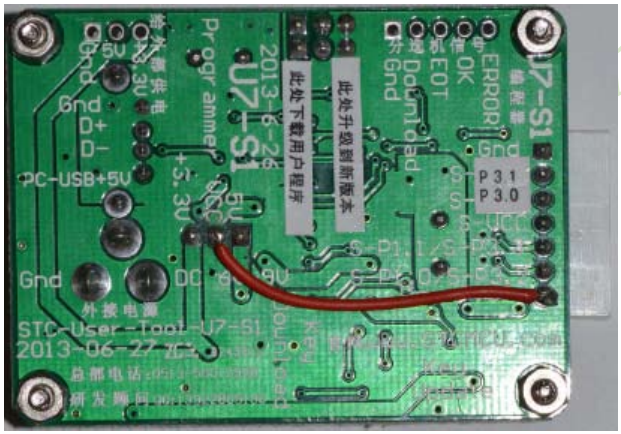


U7（5V）工具　U7（3.3V）工具　U7-S1（5V）工具　U7-S1（3.3V）工具

其反面图如下图所示：



U7反面图



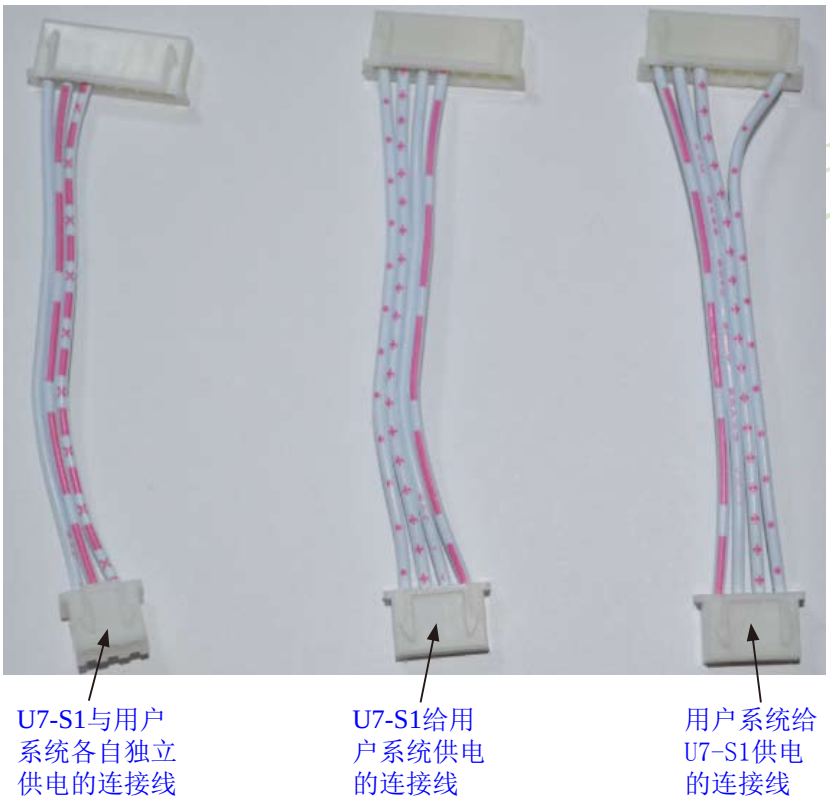
U7-S1反面图

另外还有如下的一些线材与工具相搭配使用，如：

（1）两头公的USB连接线，如下图所示：

注意：此USB线为我公司特别定制的USB加强线，可确保直接用USB供电时能够下载成功。而市面上一些比较劣质的两头公的USB线，内阻太大而导致压降很大（如USB空载时的电压为5.0V左右，当使用劣质的USB线连接U7/U7-S1，到我们的下载板上的电压可能降到4.2V或者更低，从而导致芯片处于复位状态而无法成功下载）。

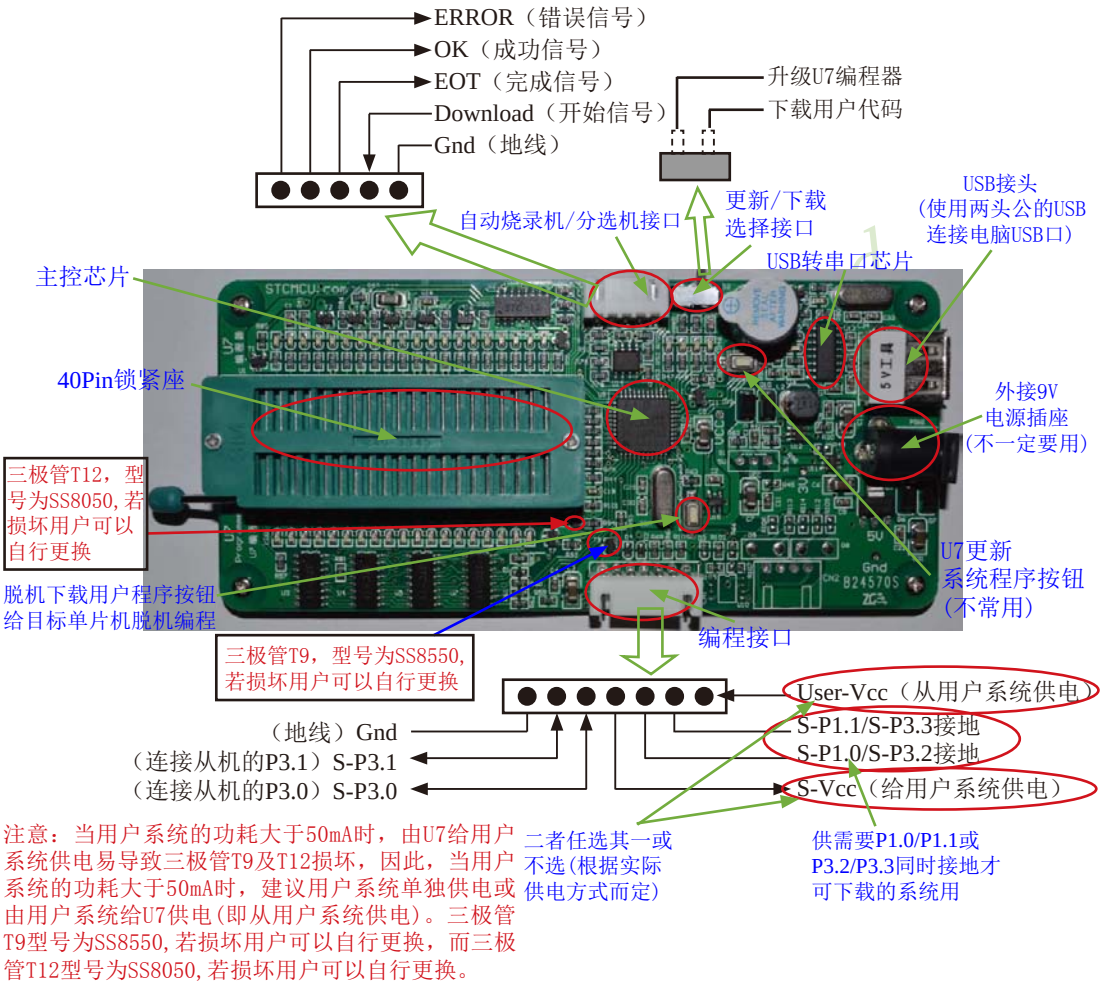
（2）U7-S1与用户系统连接的下载连接线(U7-S1与用户板上的目标单片机的连接线)，如下图所示：



15.6.1 USB型联机/脱机下载工具U7/U7-S1的功能介绍

15.6.1.1 U7的功能介绍(U7的价格为人民币100元)

下面以5V工具为例，详细介绍U7工具的各个接口及功能



编程接口：根据不同的供电方式，使用不同的下载连接线连接U7下载板和用户系统。

U7更新系统程序按钮：用于更新U7工具，当有新版本的U7固件时，需要按下此按钮对U7的主控芯片进行更新（注意：必须先将更新/下载选择接口上的跳线跳为更新U7编程器模式）。

脱机下载用户程序按钮：开始脱机下载按钮。首先PC将脱机代码下载到U7板上，然后使用下载连接线将用户系统连接到U7，再按下此按钮即可开始脱机下载（每次上电时也会立即开始下载用户代码）。

外接9V电源插座：当进行脱机下载时，可使用外部电源供电。此电源接口可支持6～9V的外部直流供电。

USB接头：用于连接电脑。使用标配的两头公的USB线连接电脑，可对目标芯片进行在线联机下载，也可使用USB供电进行脱机下载。

更新/下载选择接口：当需要对U7的底层固件进行升级时，需要将此跳线跳为更新U7编程器模式，否则，必须将跳线跳为下载用户代码模式。（跳线连接方式请参考上图）

自动烧录机/分选机接口：是用于控制自动烧录机/分选机进行自动生产的控制接口

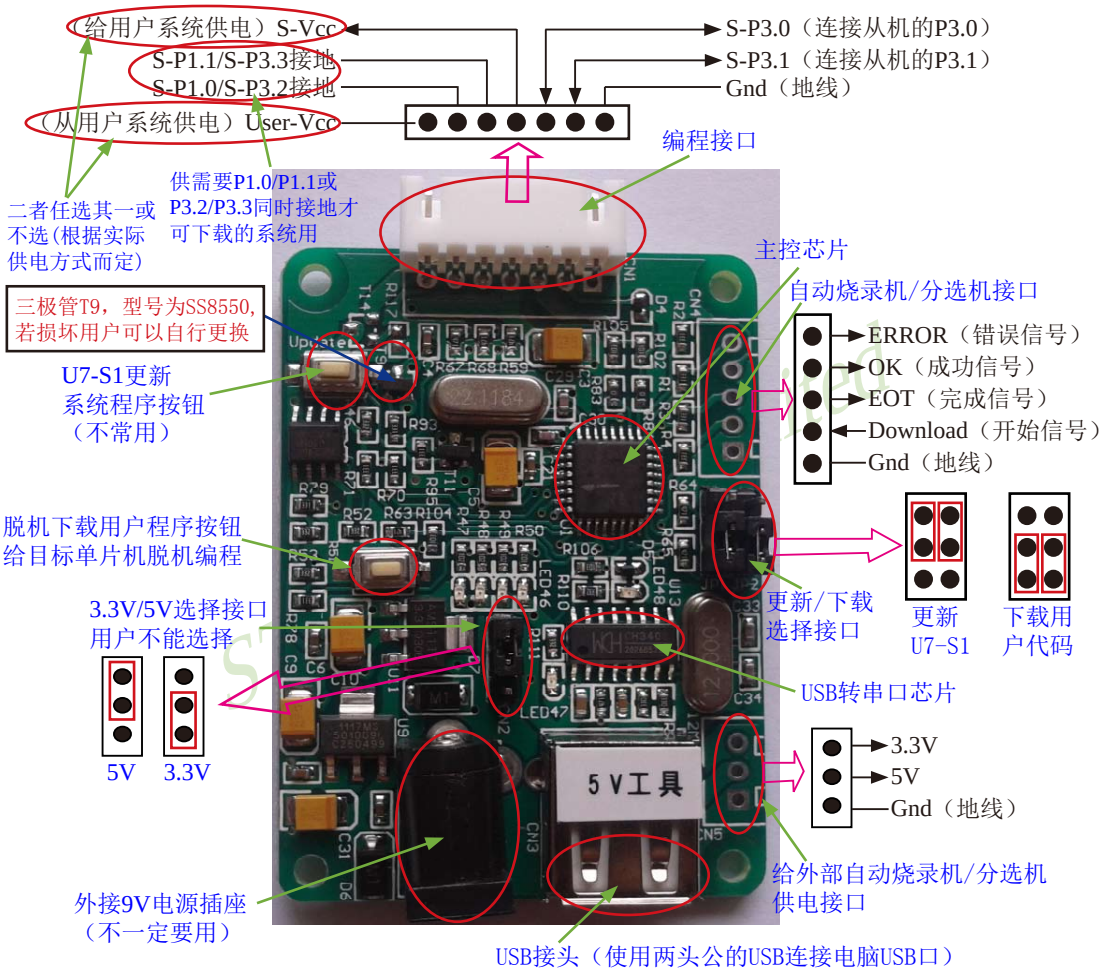
对于USB型联机/脱机下载工具，STC公司特别制作了配套的外壳，并在该外壳上对U7上相应位置的器件或管脚进行了标注，将U7安装于该外壳内（如下图所示），可对U7上的线路图一目了然。

U7安装外壳的正面图

U7安装外壳的反面图

15.6.1.2 U7-S1的功能介绍(U7-S1的价格为人民币50元)

下面以5V工具为例，详细介绍U7-S1工具的各个接口及功能



编程接口：根据不同的供电方式，使用不同的下载连接线连接U7-S1下载板和用户系统。

U7-S1更新系统程序按钮：用于更新U7-S1工具，当有新版本的U7-S1固件时，需要按下此按钮对U7-S1的主控芯片进行更新（注意：必须先将更新/下载选择接口上的跳线跳为更新模式）。

脱机下载用户程序按钮：开始脱机下载按钮。首先PC将脱机代码下载到U7-S1板上，然后使用下载连接线将用户系统连接到U7-S1，再按下此按钮即可开始脱机下载。

5V/3.3V选择接口：用于选择U7-S1下载板系统的工作电压，需要与主控芯片的工作电压一致，现版本，用户不能选，因为现主控芯片不是宽压的。（3.3V和5V的跳线选择参考上图）。

外接9V电源插座：当进行脱机下载时，可使用外部电源供电。此电源接口可支持6～9V的外部直流供电。

USB接头：用于连接电脑。使用标配的两头公的USB线连接电脑，可对目标芯片进行在线联机下载，也可使用USB供电进行脱机下载。

给外部自动烧录机/分选机供电接口：U7-S1额外提供的一个给外部供电的接口,可供应3.3V和5V

更新/下载选择接口：当需要对U7-S1的底层固件进行升级时，需要将此跳线跳为更新U7-S1模式，否则，必须将跳线跳为下载用户代码模式。（跳线连接方式请参考上图）

自动烧录机/分选机接口：是用于控制自动烧录机/分选机进行自动生产的控制接口

注意：当用户系统的功耗大于50mA时，由U7-S1给用户系统供电易导致三极管T9(如上图所示)损坏，因此，当用户系统的功耗大于50mA时，建议用户系统单独供电或由用户系统给U7-S1供电(即从用户系统供电)。三极管T9型号为SS8550,若损坏用户可以自行更换。

由上述两小节可知，USB型联机/脱机下载工具U7与U7-S1相比，只是多了一个“编程锁紧座”，其他各模块及功能均相同，为避免赘述，下文“USB型联机/脱机下载工具的使用介绍”中均仅以USB型联机/脱机下载工具U7为例，用户可对照U7的使用介绍，使用U7-S1的相应模块。

STC MCU Limited

15.6.2 如何安装USB型联机/脱机下载工具U7/U7-S1的驱动程序

U7/U7-S1 下载板上使用了一颗CH340 的USB转串口通用芯片。这样可以省去部分没有串口的电脑必须额外买一条USB转串口线才可下载的麻烦。但CH340 和其它USB转串口线一样，在使用之前必须先安装驱动程序。驱动程序可以进行手动安装，也可以自动安装。

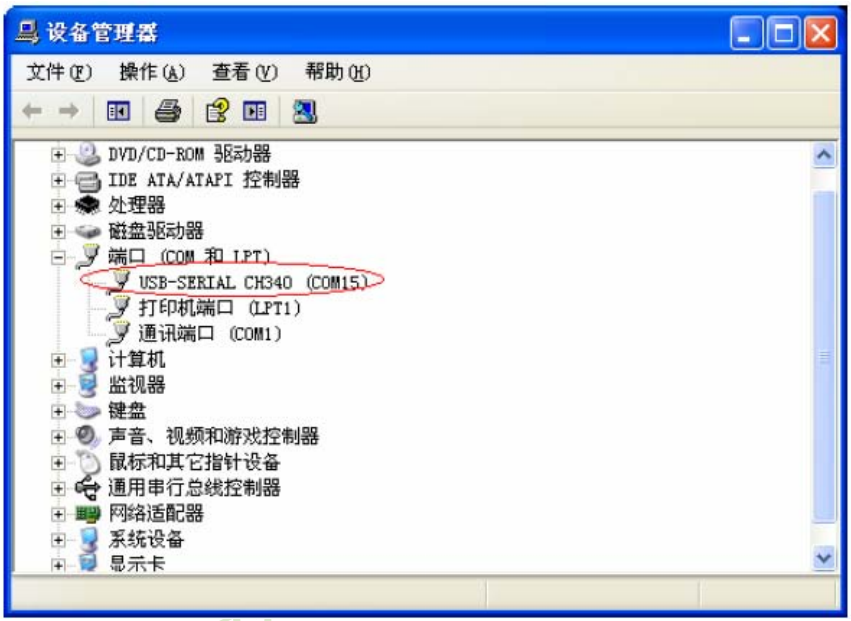
1、手动安装USB型联机/脱机下载工具U7/U7-S1的驱动程序

在STC的官方网站上手动下载驱动程序，驱动的下载链接为：U7编程器USB转串口驱动（http://www.stcmcu.com/STCISP/CH341SER.exe）。网站上的驱动地址如下图所示：

驱动程序下载到本机后，直接双击可执行程序并运行，出现下图所示的界面，点击“安装”按钮开始自动安装驱动

直至弹出下面的画面表示驱动已成功安装

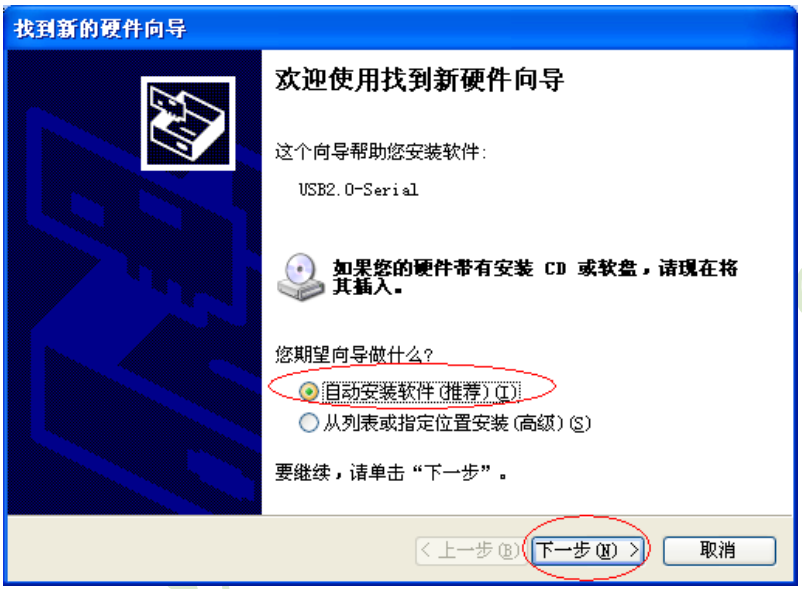
然后使用 STC 提供的两头公 USB 连接线将 U7/U7-S1 下载板连接到电脑，打开电脑的设备管理器，在端口设备类下面，如果有类似“USB-SERIAL CH340 (COMx)”的设备，就表示 U7/U7-S1 可以正常使用了。如下图所示（不同的电脑，串口号可能会不同）



注意：在后面使用STC-ISP下载软件时，选择的串口号必须选择与此相对应的串口号，如下图所示

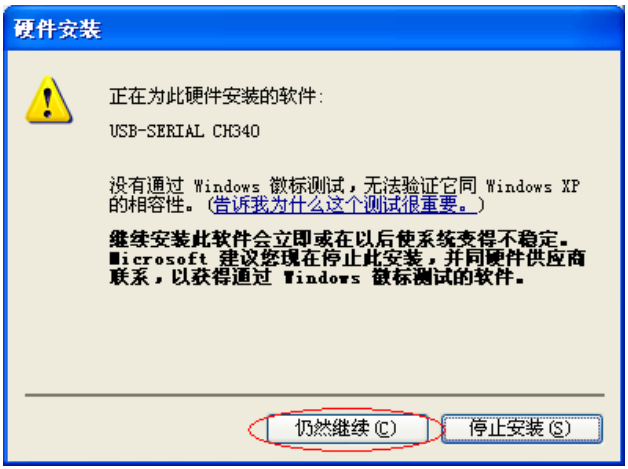
2、自动安装USB型联机/脱机下载工具U7/U7-S1的驱动程序

若用户所使用的 STC-ISP 下载软件为 V6.58C 及以上版本，则打开该 STC-ISP 下载软件时软件会自动检测本机的 U7/U7-S1 驱动程序的安装情况，若没有安装驱动程序，软件会自动将相应的驱动程序复制到系统目录，此时拔出上一次插入的 U7/U7-S1 工具并再次将其插上时，会出现如下提示框：



选“自动安装软件(推荐)(I)”选项，并点击【下一步】按钮，会出现如下画面：

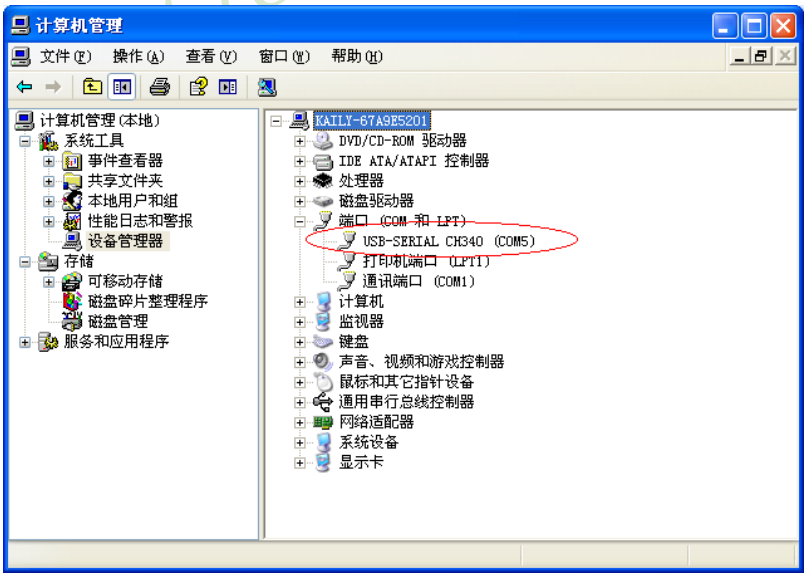
在接下来出现的如下面的对话框中，选中【仍然继续】：



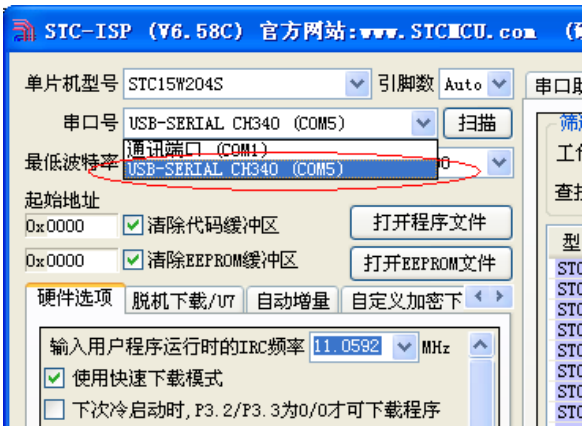
系统便会开始自动安装驱动，如下图所示：

直至出现如下画面，点击【完成】按钮：

至此，U7/U7-S1的驱动程序便自动安装完成了。如手动安装驱动程序一样，也会如下图所示（不同的电脑，串口号可能会不同）：



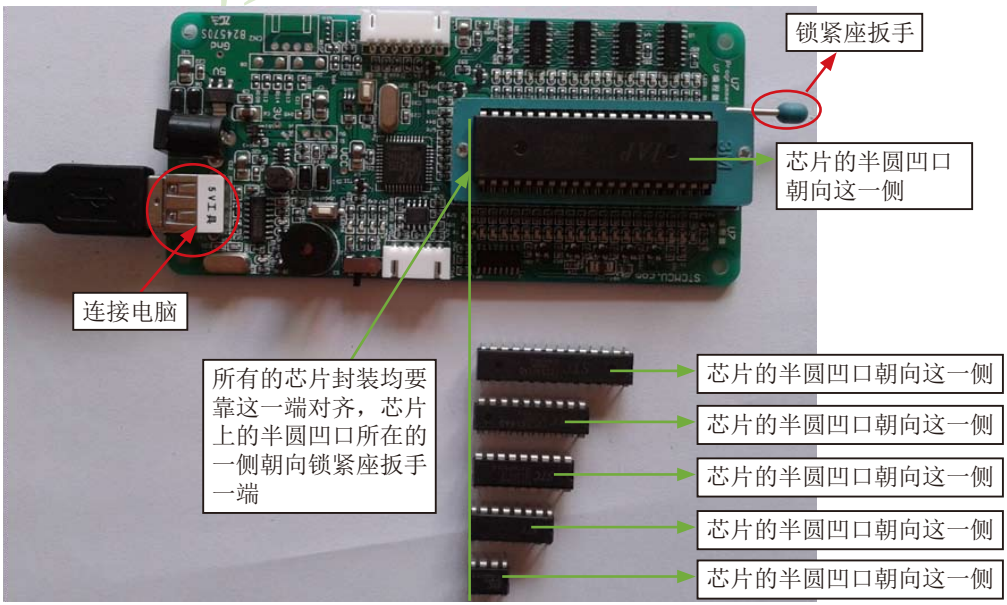
注意：在后面使用STC-ISP下载软件时，选择的串口号必须选择与此相对应的串口号，如下图所示：



15.6.3 U7/U7-S1的在线联机下载使用说明

15.6.3.1 目标芯片直接安装于U7上的在线联机下载使用说明

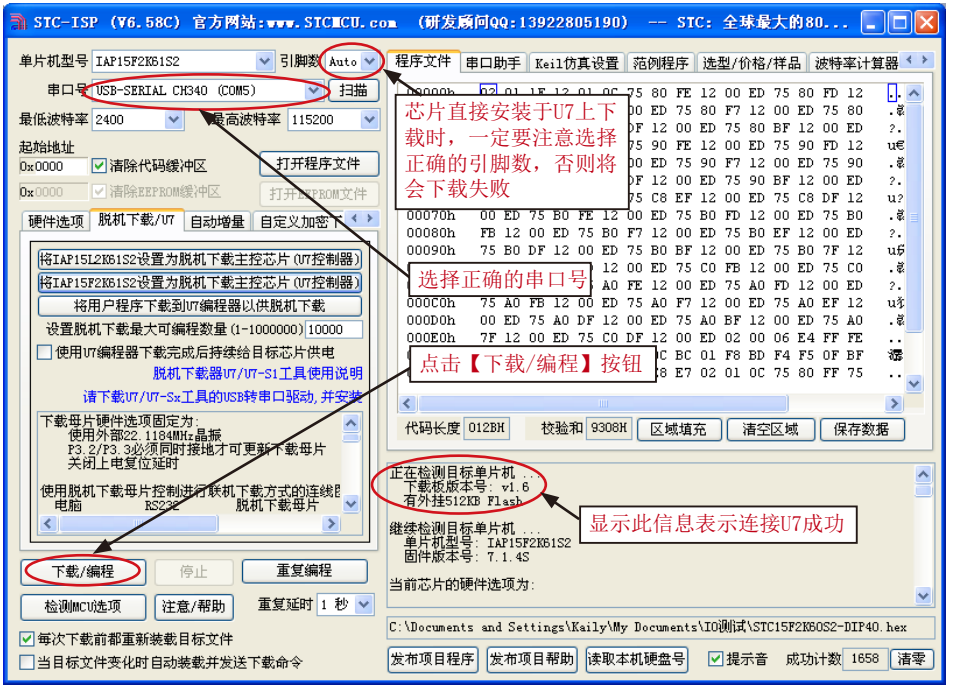
首先使用STC提供的两头公USB连接线将U7连接电脑，再将目标单片机按如下图所示的方向安装在U7上：



STC MCU Limited

然后在用STC-ISP下载软件下载程序时，在STC-ISP下载软件中选择正确的串口号(USB转串口扩展的)，点击【下载/编程】按钮即可开始在线下载。

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.58C).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。

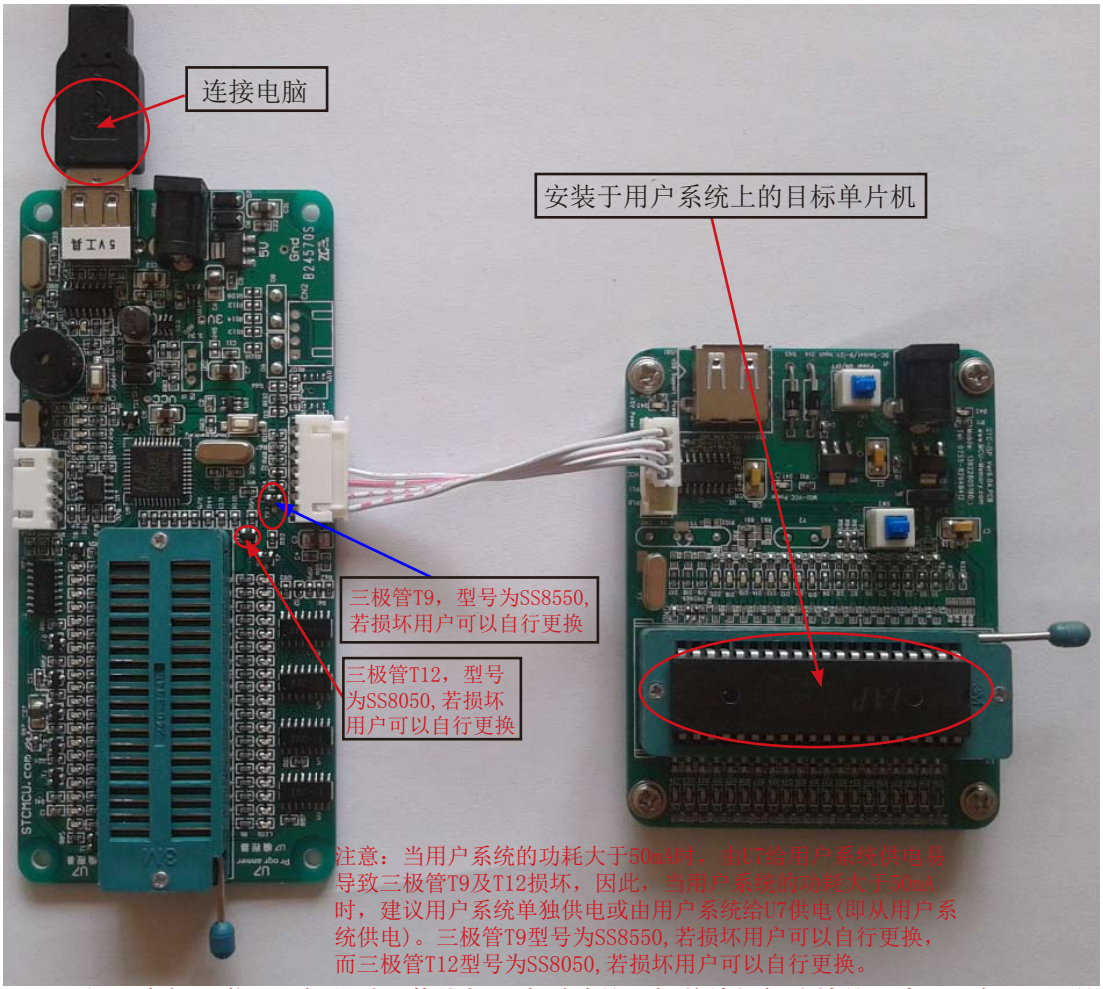


当信息框中有输出下载板的版本号信息以及外挂Flash的相应信息时，表示已正确检测到U7下载工具。

下载的过程中，U7下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

15.6.3.2 通过下载线连接目标芯片的U7/U7-S1的在线联机下载使用说明

首先使用STC提供的两头公USB连接线连接电脑，再将U7/U7-S1通过下载线与用户系统的目标单片机相连接，连接方式如下图所示：



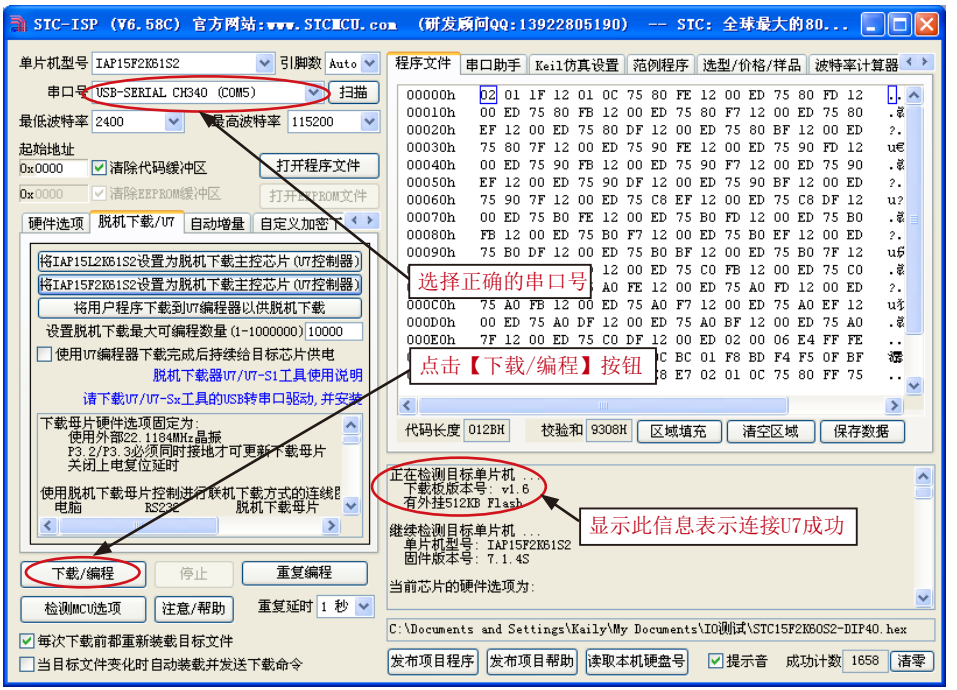
虽然图中仅列举了U7与通过下载线与用户系统的目标单片机相连接的示意图，但U7-S1的连接方式与该图相同，因此不再赘述。

然后在用STC-ISP下载软件下载程序时，在STC-ISP下载软件中选择正确的串口号(USB转串口扩展的)，点击【下载/编程】按钮即可开始在线下载。

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.58C).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。

然后在用STC-ISP下载软件下载程序时，在STC-ISP下载软件中选择正确的串口号(USB转串口扩展的)，点击【下载/编程】按钮即可开始在线下载。

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.58C).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。



当信息框中有输出下载板的版本号信息以及外挂Flash的相应信息时，表示已正确检测到U7/U7-S1下载工具。

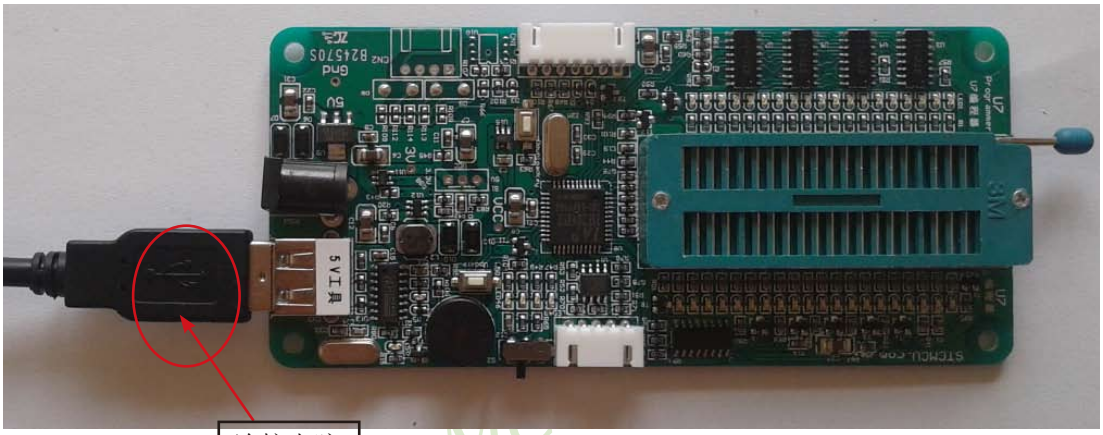
下载的过程中，U/U7-S17下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

15.6.4 U7/U7-S1的脱机下载使用说明

15.6.4.1 目标芯片直接安装于U7上时通过USB供电进行脱机下载的使用说明

使用USB给U7从而进行脱机下载的步骤如下：

（1）使用STC提供的两头公USB连接线将U7下载板连接到电脑，如下图：

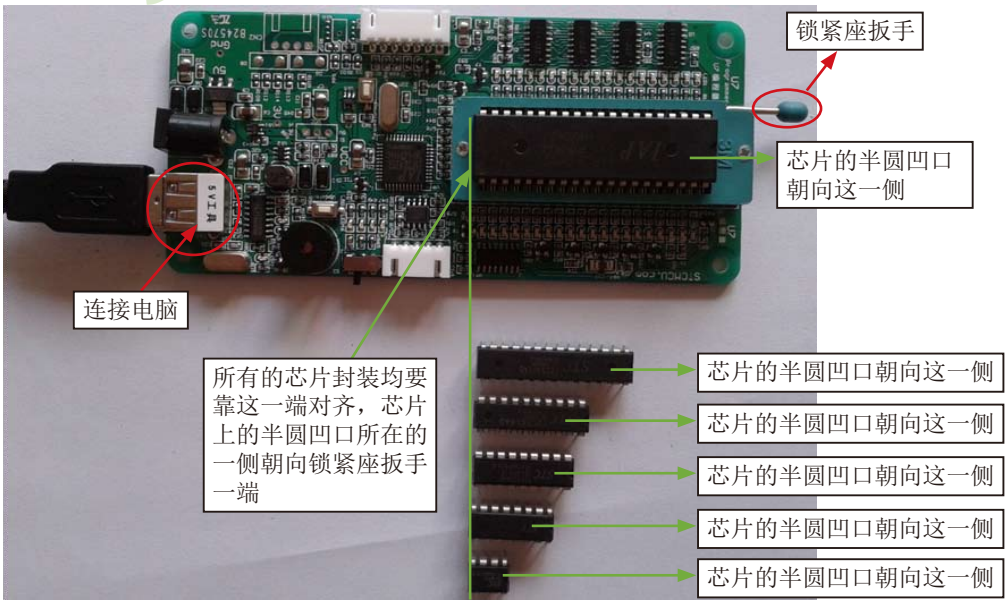


（2）在ISP下载程序“STC-ISP (V6.65).exe”中按如下图所示的步骤进行设置：

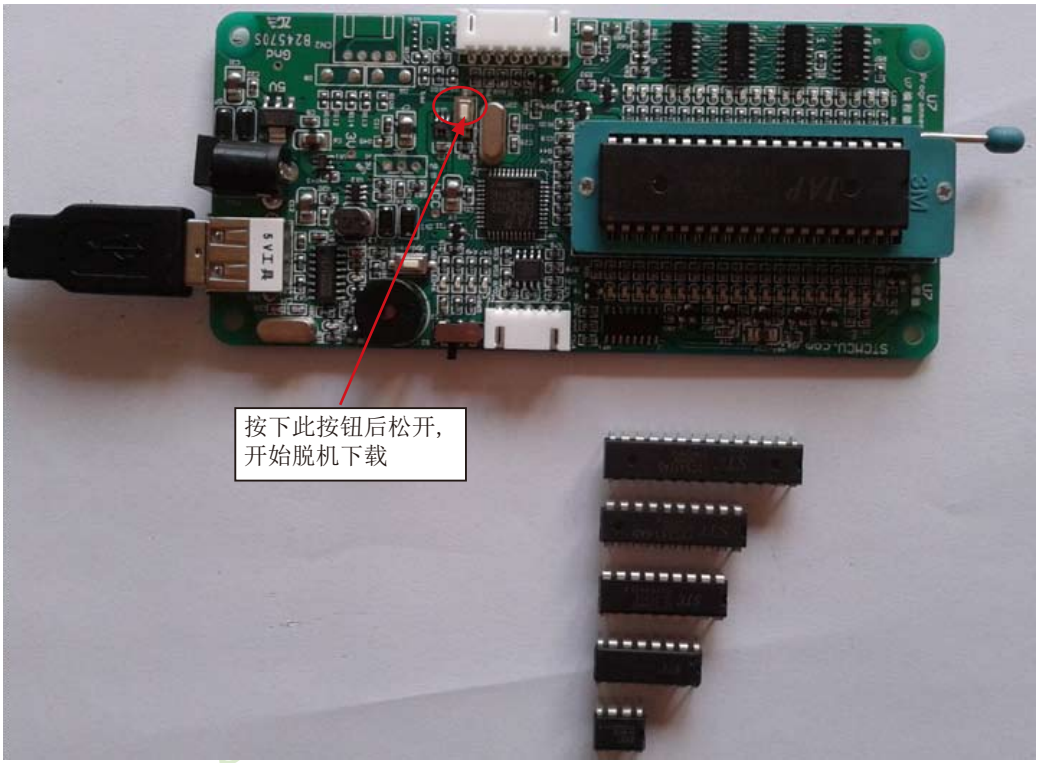
建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.58C).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到U7下载工具中。

（3）再将目标单片机如下图所示的方向放在U7下载工具，如下图所示



（4）然后按下如下图所示的按钮后松开，即可开始脱机下载：

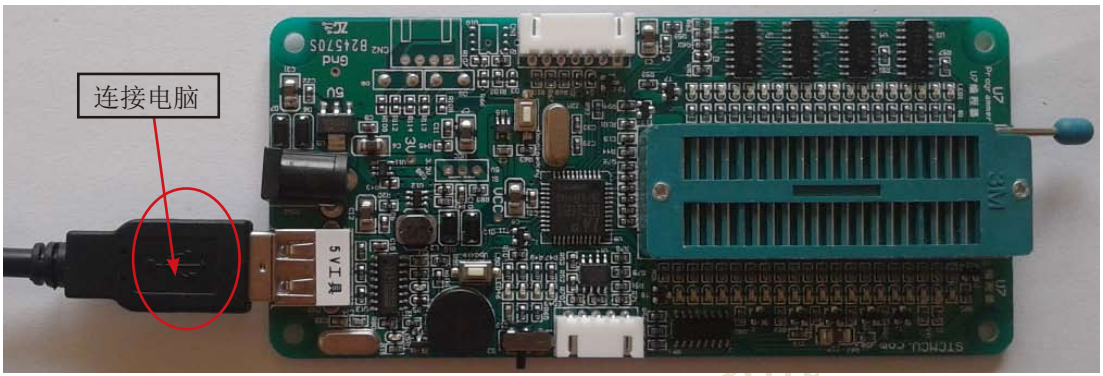


下载的过程中，U7下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

15.6.4.2 目标芯片通过下载线连接于U7/U7-S1时用USB供电进行脱机下载的使用说明

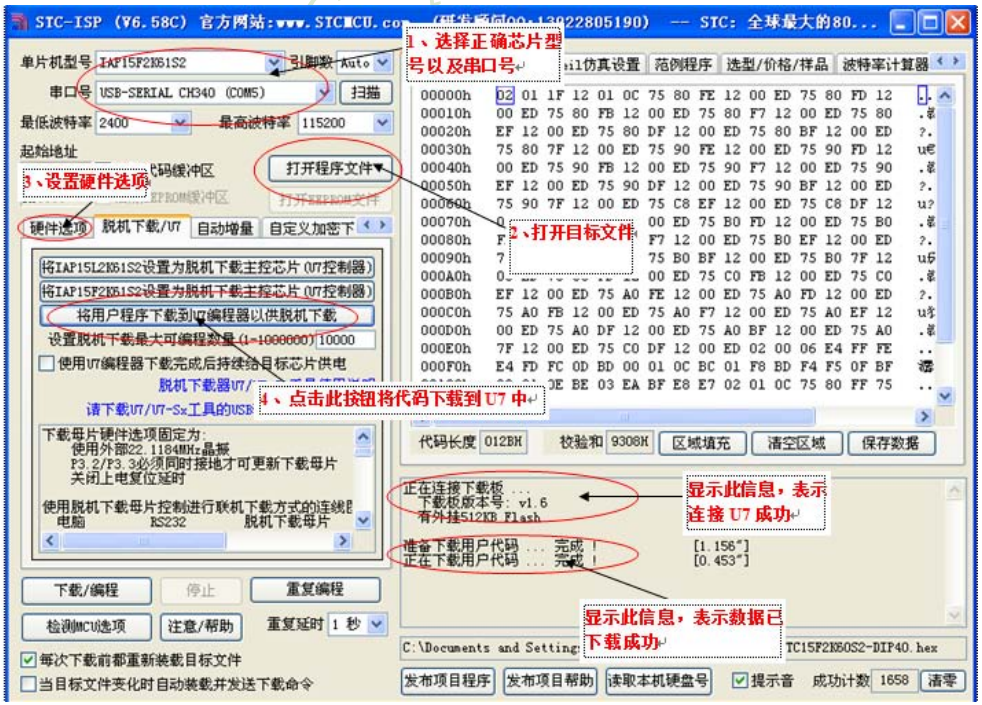
使用USB给U7/U7-S1从而进行脱机下载的步骤如下：

（1）使用STC提供的两头公USB连接线将U7/U7-S1下载板连接到电脑，如下图：



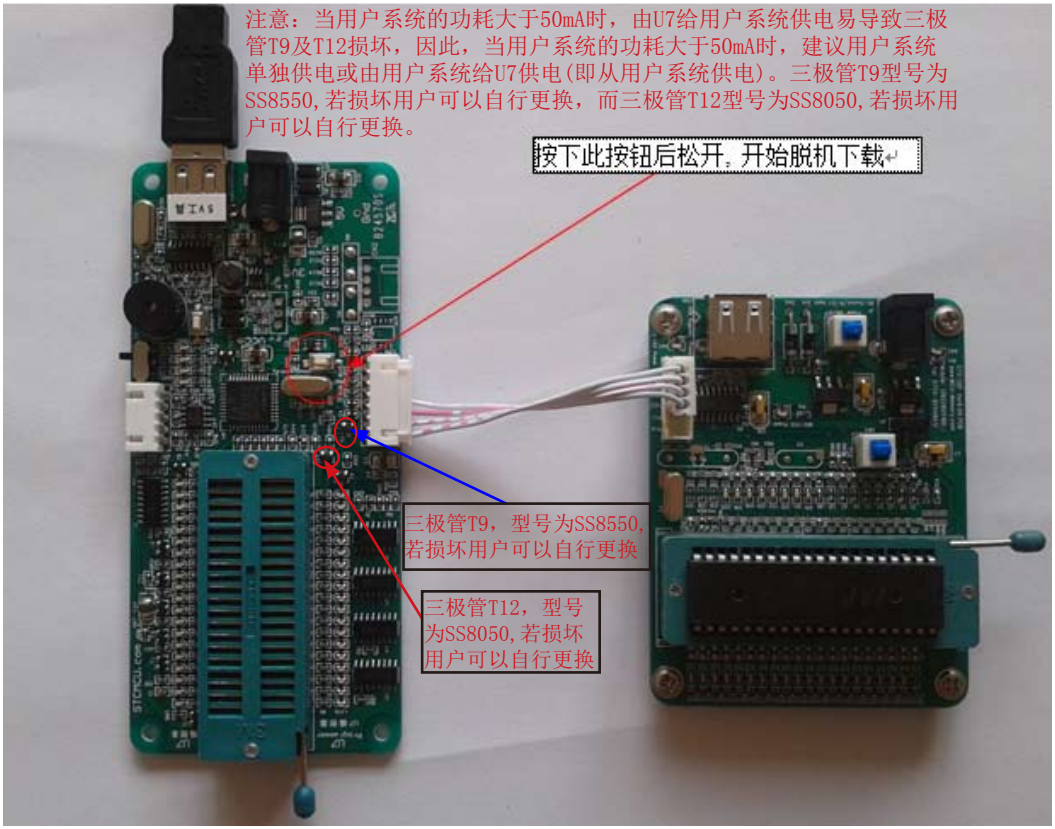
（2）在ISP下载程序“STC-ISP (V6.65).exe”中按如下图所示的步骤进行设置：

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.58C).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到U7下载工具中。

（3）然后使用连接线连接电脑、将U7/U7-S1下载工具以及用户系统（目标单片机）如下图所示的方式连接起来，并按下图所示的按钮后松开，即可开始脱机下载

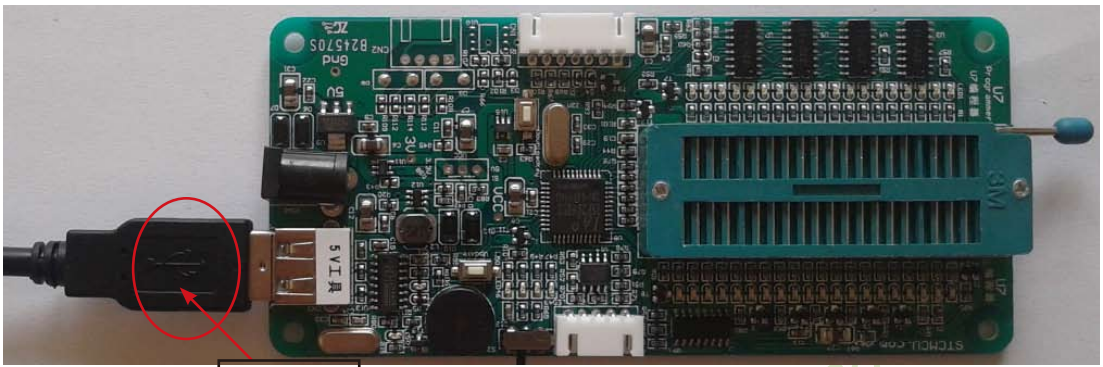


下载的过程中，U7/U7-S1下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

本节附图中虽然仅列举了U7的连接示意图，但U7-S1的连接方式与U7相同，因此不再赘述。

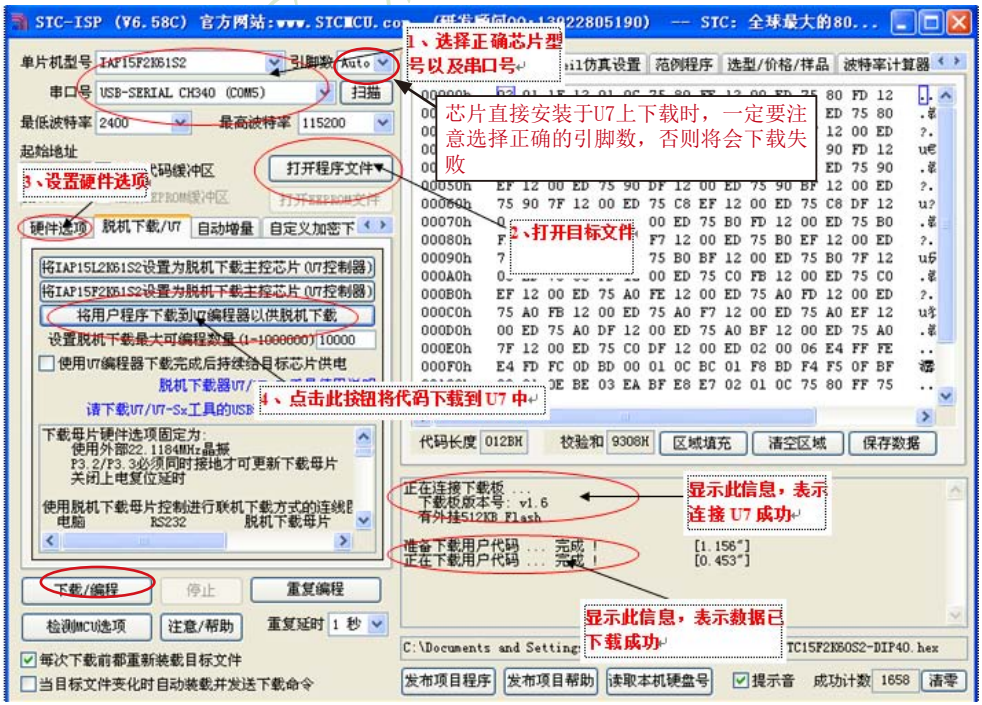
15.6.4.3 目标芯片直接安装于U7上时通过外部9V电源供电进行脱机下载的使用说明

（1）首先使用STC提供的两头公USB连接线将U7下载板连接到电脑，如下图：



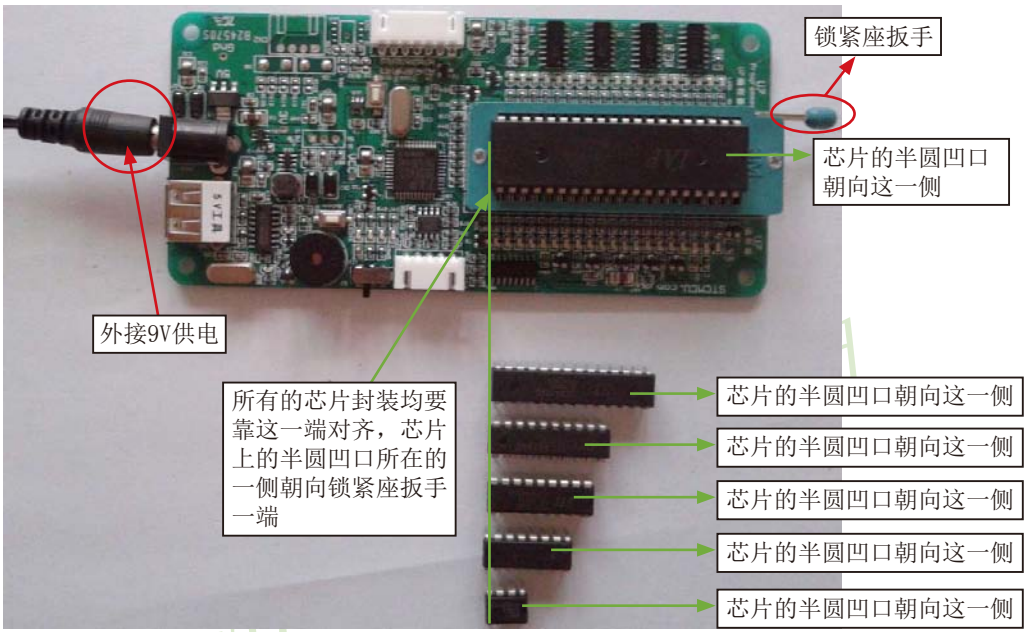
（2）在ISP下载程序“STC-ISP (V6.65).exe”中按如下图所示的步骤进行设置：

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.58C).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。

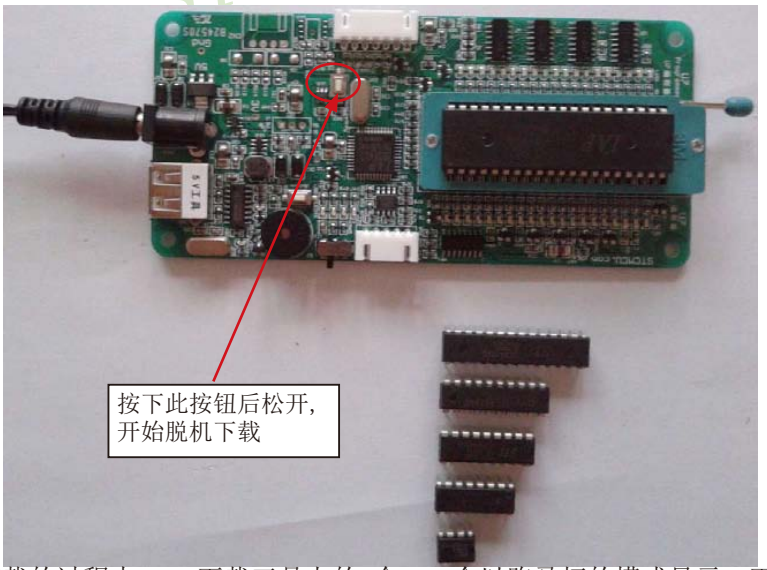


按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到U7下载工具中。

（3）再将目标单片机如下图所示的方向放在U7脱机下载工具，如下图所示



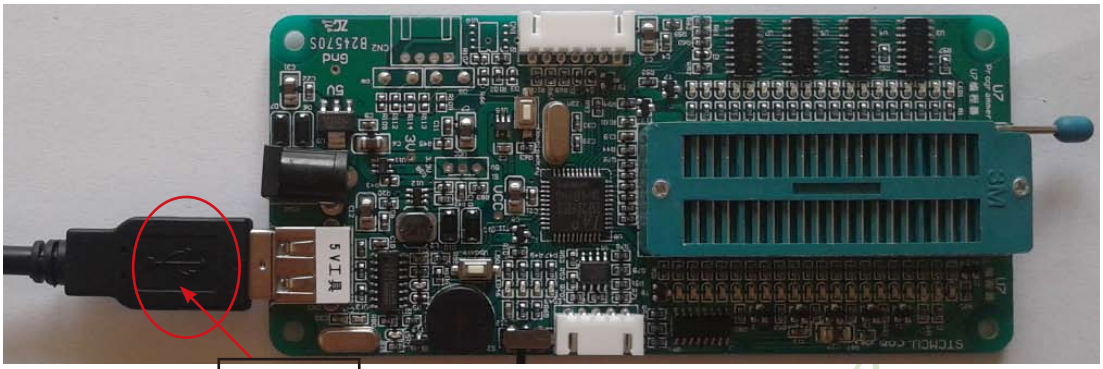
（4）然后按下如下图所示的按钮后松开，即可开始脱机下载



下载的过程中，U7下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

15.6.4.4 目标芯片通过下载线连接于U7/U7-S1时用外部9V电源供电进行脱机下载的说明

（1）首先使用STC提供的两头公USB连接线将U7/U7-S1下载板连接到电脑，如下图：

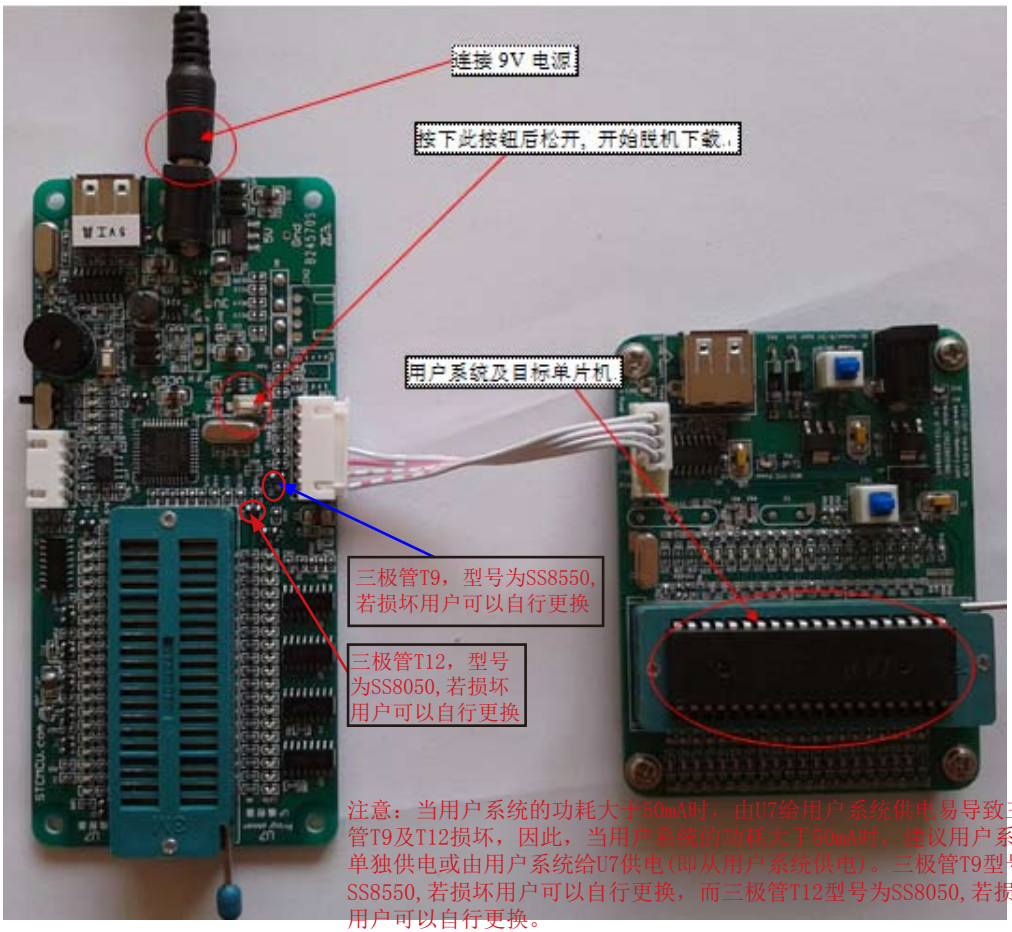


（2）在ISP下载程序“STC-ISP (V6.65).exe”中按如下图所示的步骤进行设置：

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.58C).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到U7下载工具中

（3）然后按下图所示的方式连接9V电源，并按下图中所示按钮后松开，即可开始脱机下载：

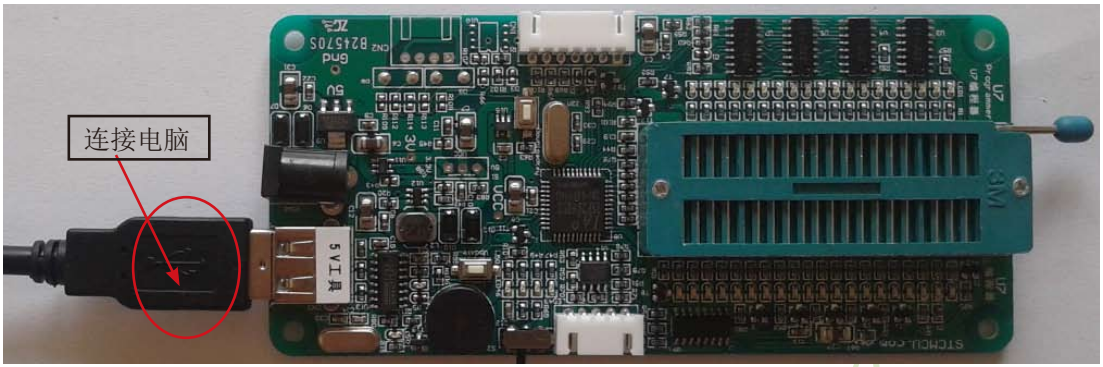


下载的过程中，U7/U7-S1下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

本节附图中虽然仅列举了U7的连接示意图，但U7-S1的连接方式与U7相同，因此不再赘述。

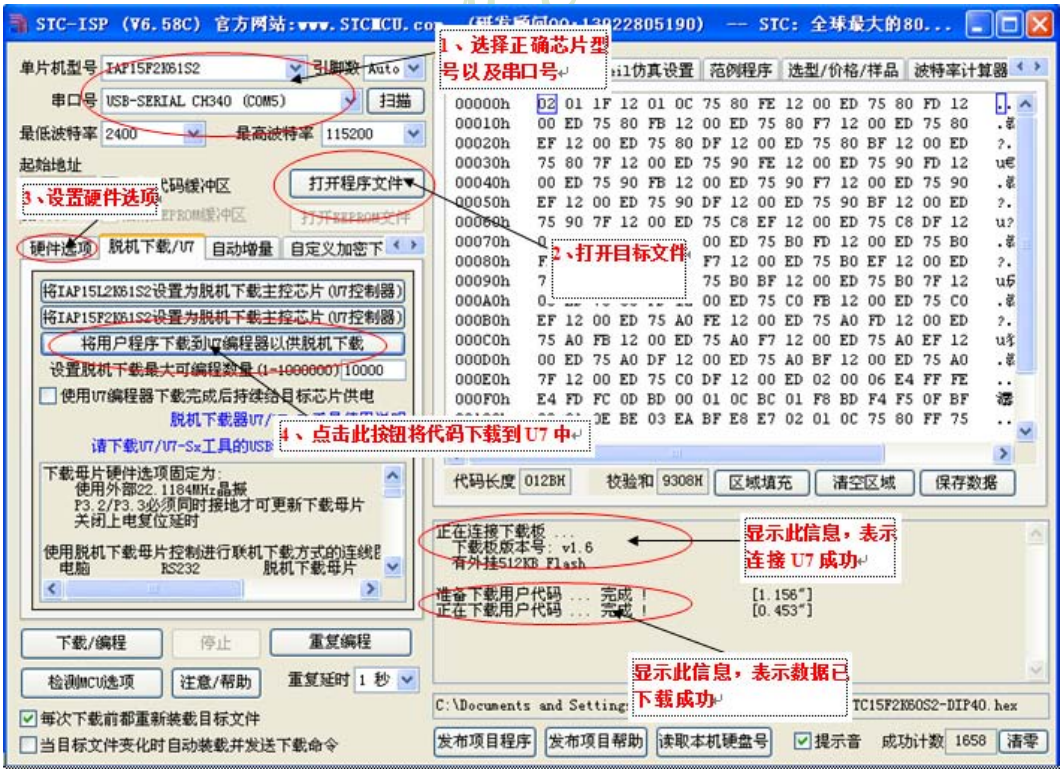
15.6.4.5 通过用户板给简易脱机下载器U7/U7-S1供电进行脱机下载的使用说明

（1）首先使用STC提供的两头公USB连接线将U7/U7-S1下载板连接到电脑，如下图：



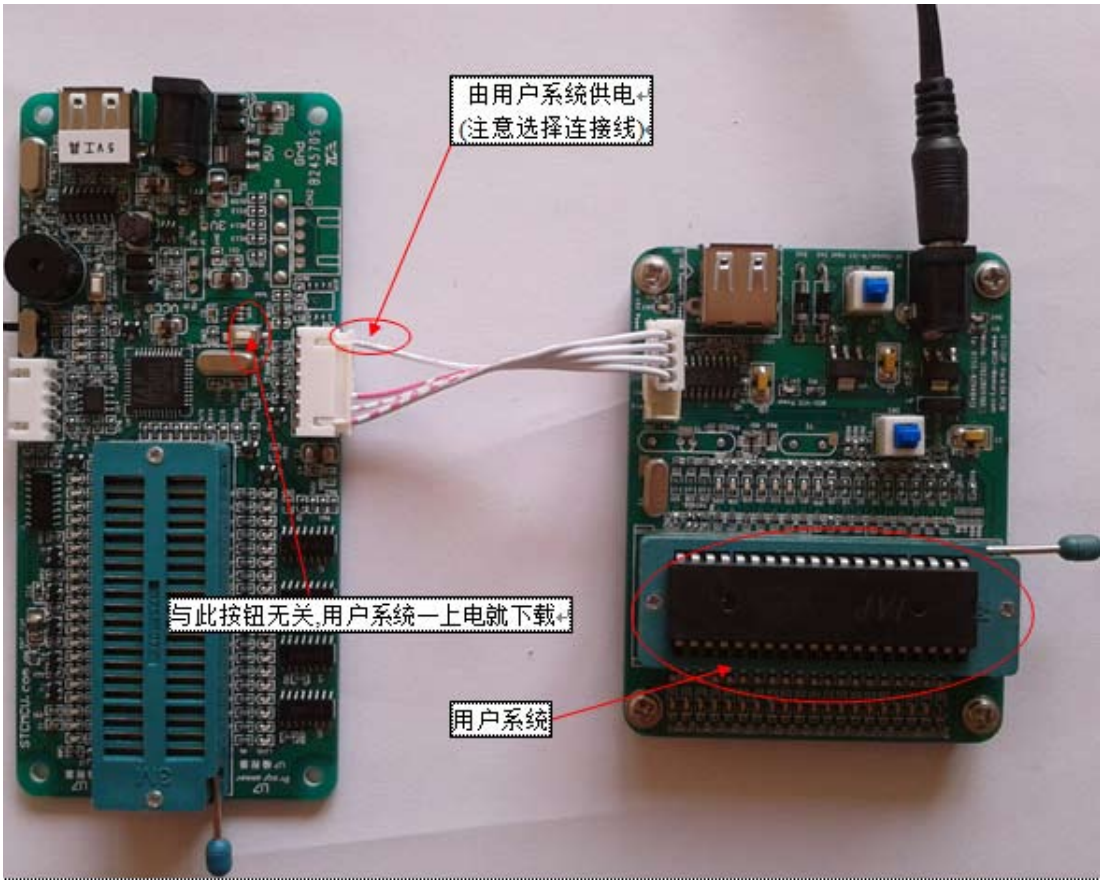
（2）在ISP下载程序“STC-ISP (V6.65).exe”中按如下图所示的步骤进行设置：

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.58C).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到U7下载工具中

（3）然后按下图所示的方式连接U7/U7-S1与用户系统，并按下图中所示按钮后松开，即可开始脱机下载：

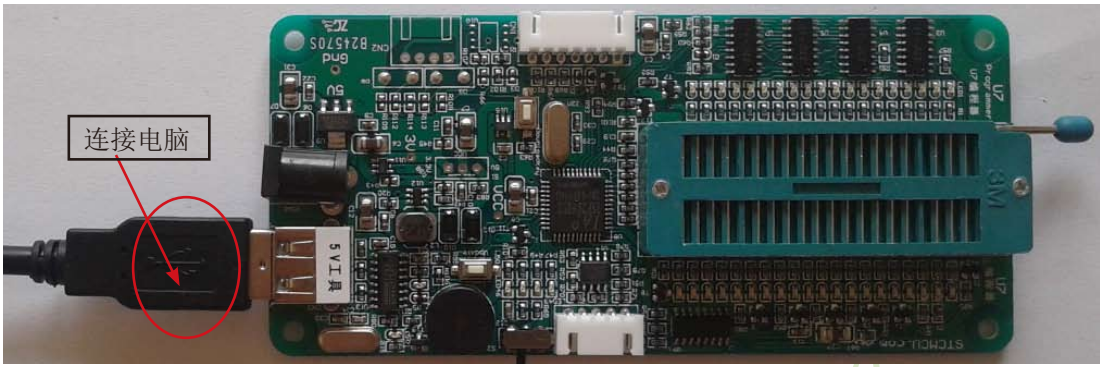


下载的过程中，U7/U7-S1下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

本节附图中虽然仅列举了U7的连接示意图，但U7-S1的连接方式与U7相同，因此不再赘述。

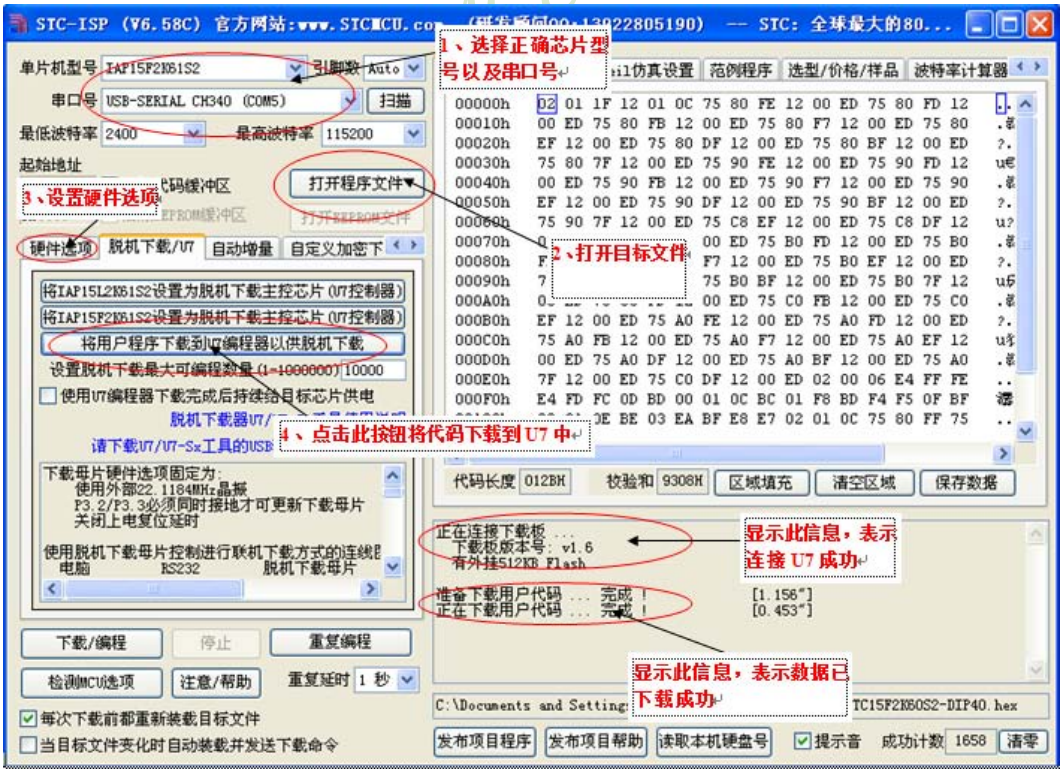
15.6.4.6 U7/U7-S1与用户系统各自独立供电进行脱机下载的使用说明

（1）首先使用STC提供的两头公USB连接线将U7/U7-S1下载板连接到电脑，如下图：



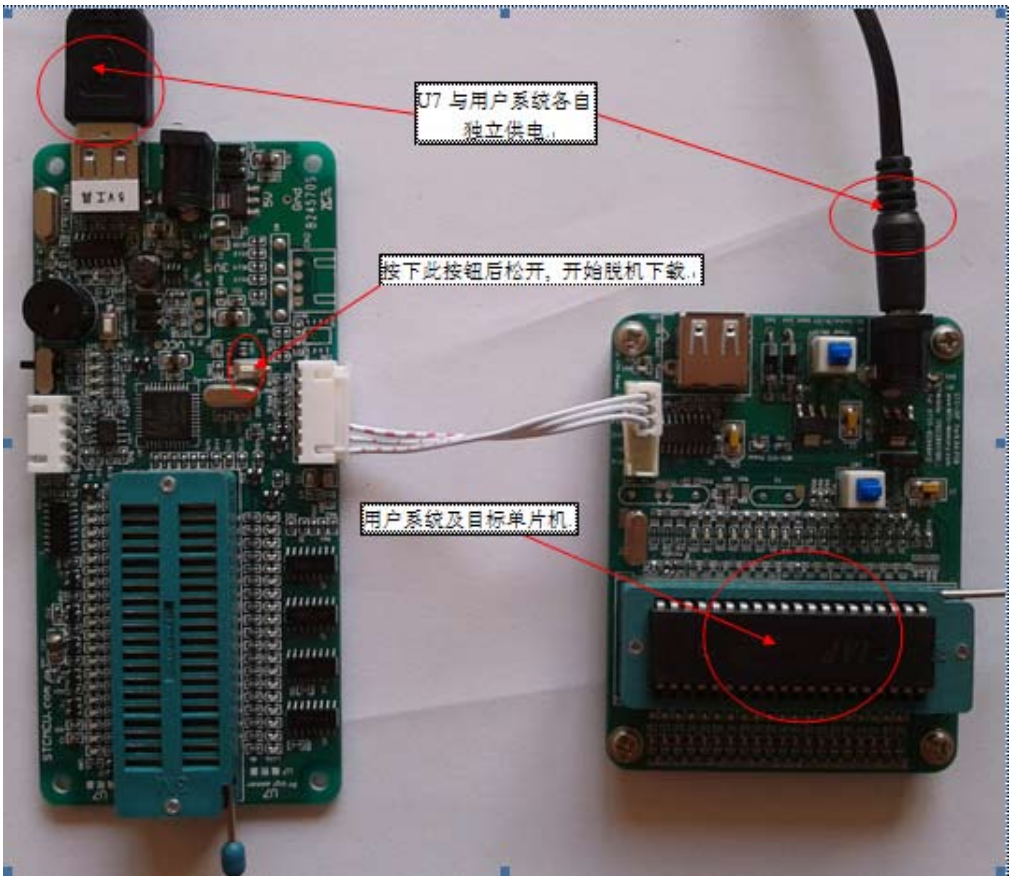
（2）在ISP下载程序“STC-ISP (V6.65).exe”中按如下图所示的步骤进行设置：

建议用户用最新版本的STC-ISP下载软件“STC-ISP (V6.58C).exe”（请随时留意STC官方网站http://www.STCMCU.com中STC-ISP下载软件的更新，强烈建议用户在官方网站http://www.STCMCU.com中下载最新版本的软件使用）。



按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到U7/U7-S1下载工具中

（3）然后按下图所示的方式连接U7/U7-S1与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



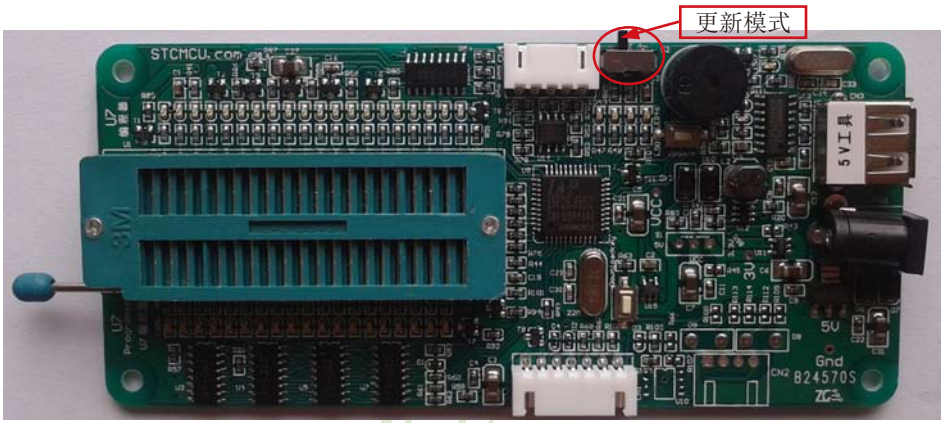
下载的过程中，U7/U7-S1下载工具上的4个LED会以跑马灯的模式显示。下载完成后，若下载成功，则4个LED会同时亮、同时灭；若下载失败，则4个LED全部不亮。

本节附图中虽然仅列举了U7的连接示意图，但U7-S1的连接方式与U7相同，因此不再赘述。

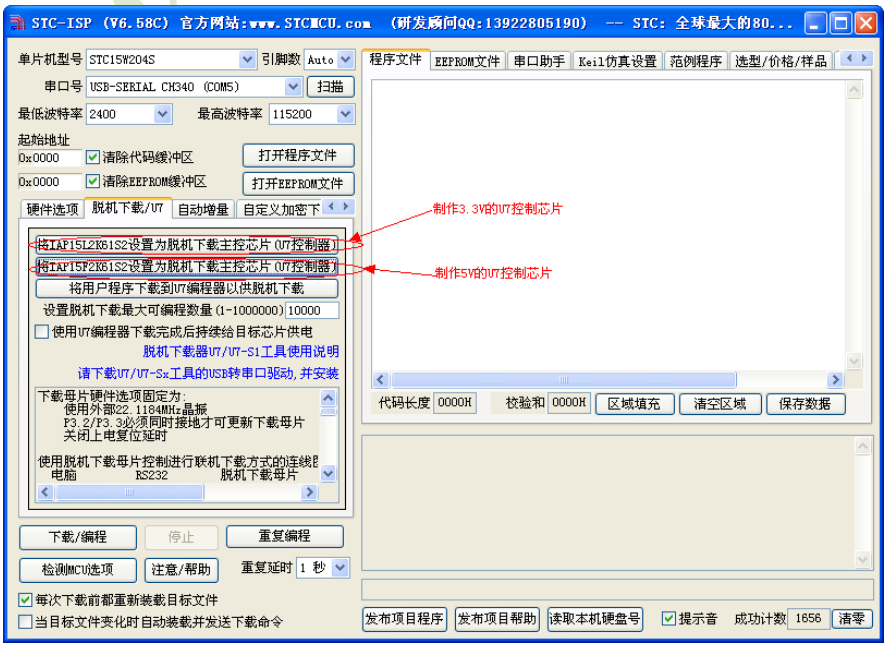
15.6.5 制作/更新USB型联机/脱机下载工具U7/U7-S1

15.6.5.1 制作U7/U7-S1下载母片(U7控制母片)

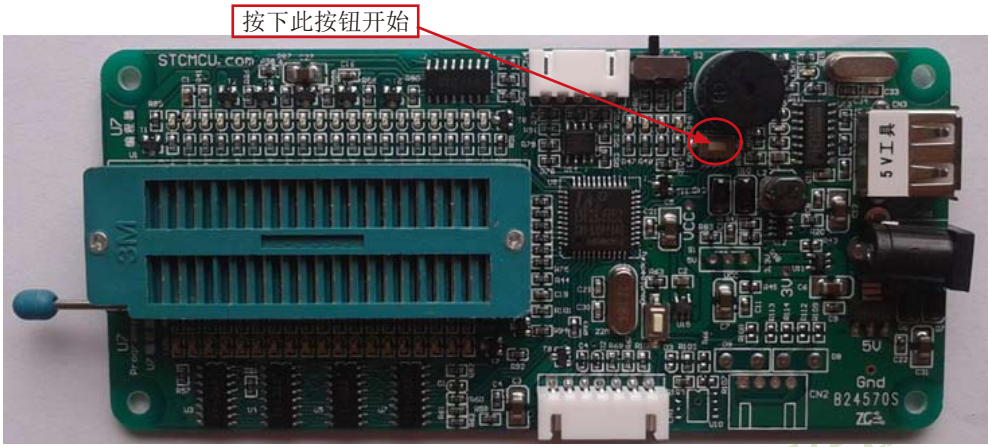
在制作U7/U7-S1下载母片之前需要将U7/U7-S1下载板的“更新/下载选择接口”设置为更新模式，如下图所示：(下图以U7为例，因U7-S1的模块与U7相同，因此在U7-S1下载板上找到相对应的“更新/下载选择接口”，同样将其设置为更新模式，下文相同)



然后在ISP下载程序“STC-ISP (V6.65).exe”中的“脱机下载/U7”页面中点击“将IAP15F2K61S2设置为脱机下载主控芯片（U7控制器）”按钮（5V下载板）或者点击“将IAP15L2K61S2设置为脱机下载主控芯片（U7控制器）”按钮（3.3V下载板），如下图：
（注意：一定要选择U7/U7-S1所对应的串口）

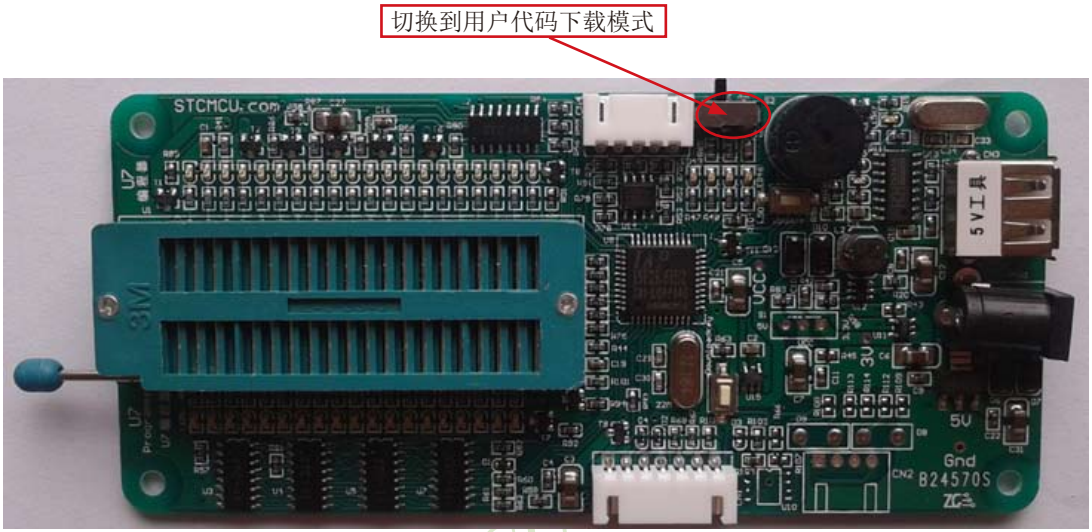


然后按下如下图所示的按键，并松开后才能开始制作U7/U7-S1下载母片：



在出现如下画面表示U7/U7-S1控制芯片制作完成：

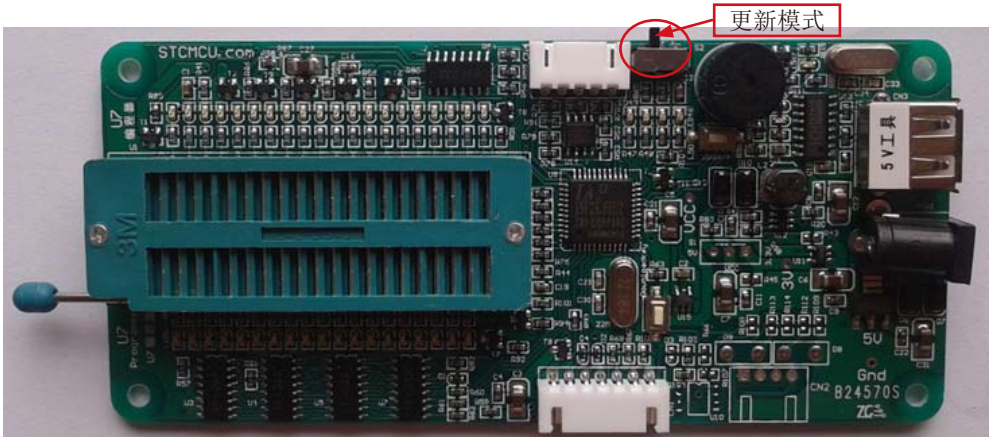
制作完成后，一定不要忘记将U7/U7-S1的“更新/下载选择接口”切换回下载用户代码模式，并将U7/U7-S1下载工具重新上电，如下图所示：（否则将不能正常进行下载）



STC MCU

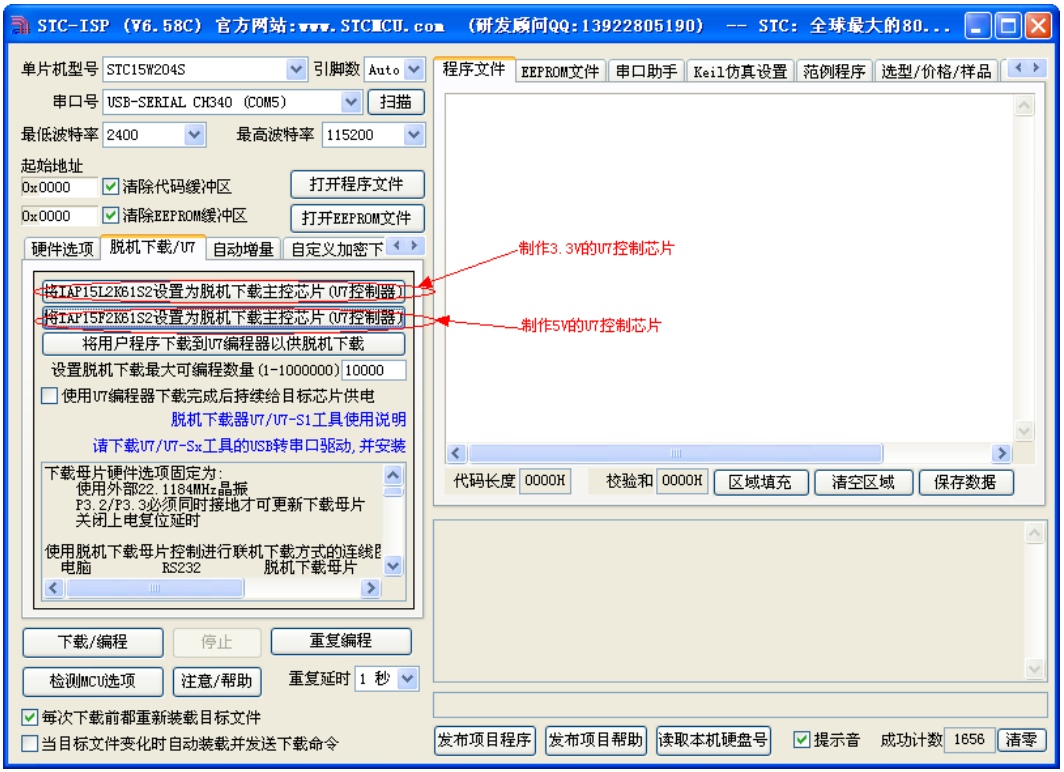
15.6.5.2 手动升级U7/U7-S1

在手动升级U7/U7-S1之前需要将“更新/下载选择接口”选中为更新模式，如下图所示：



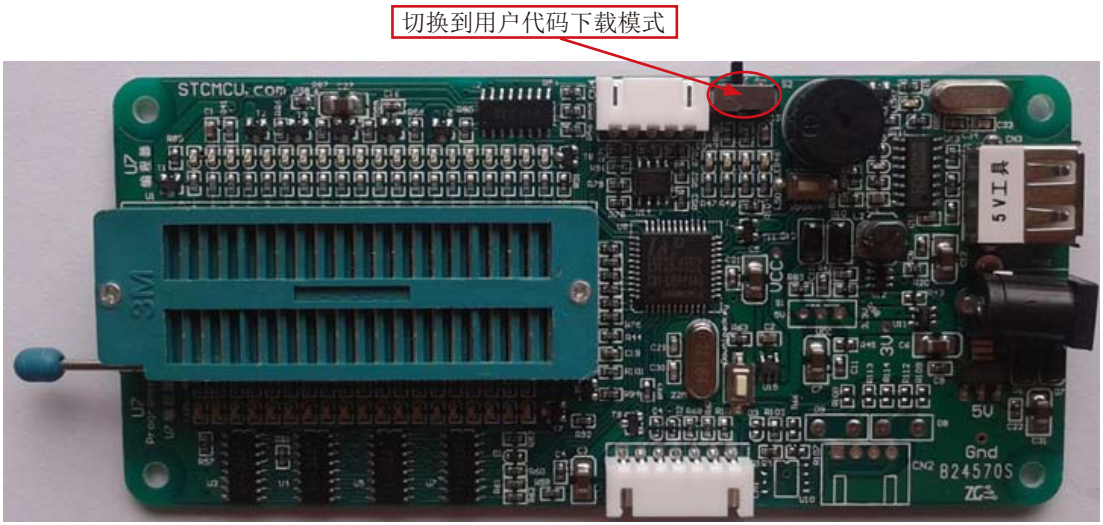
然后在ISP下载程序中的“脱机下载/U7”页面中点击“将IAP15F2K61S2设置为脱机下载主控芯片（U7控制器）”按钮（5V下载板）或者点击“将IAP15L2K61S2设置为脱机下载主控芯片（U7控制器）”按钮（3.3V下载板），如下图：

（注意：一定要选择U7/U7-S1所对应的串口）



此时由于主控芯片已经被设置为U7/U7-S1的下载母片，则点击上面对应的按钮后，芯片会自动进行更新，中间不需要按其它的按键（特殊情况：若软件一直没有反应，则需要用户手动按一下“更新/update”按钮，芯片才能进行更新），直至出现如下画面表示U7/U7-S1控制芯片升级完成：

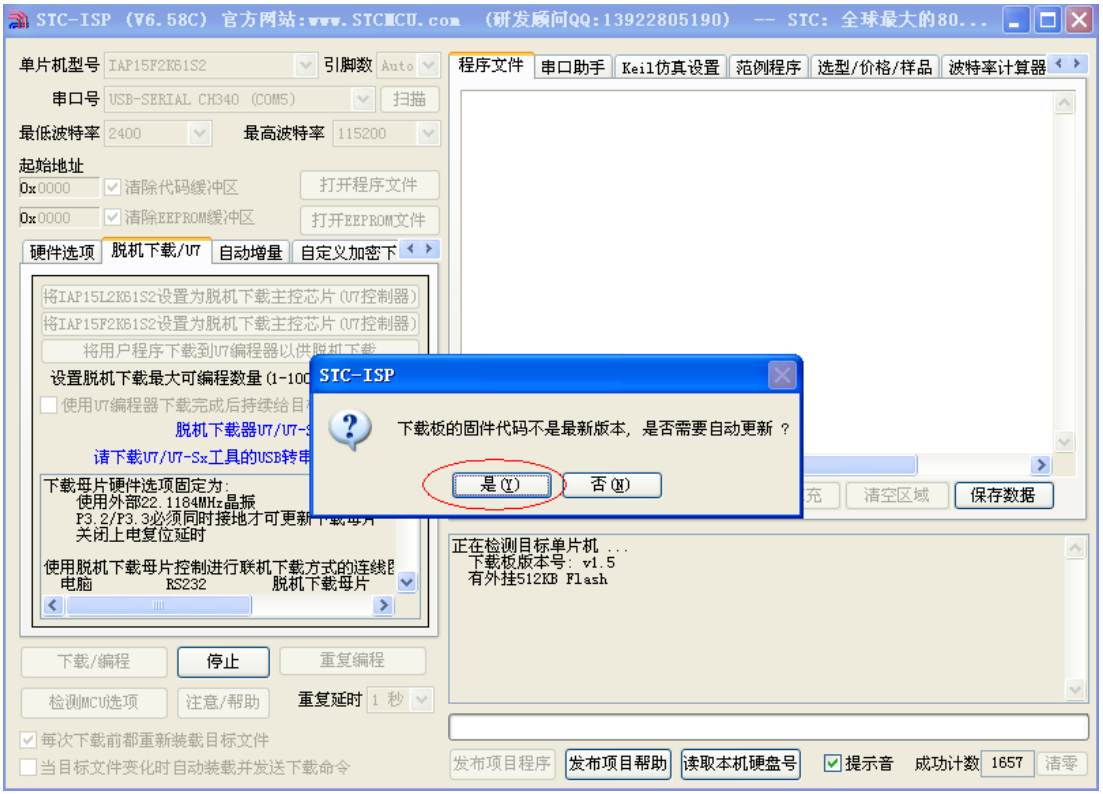
升级完成后，一定不要忘记将U7/U7-S1的“更新/下载选择接口”切换回下载用户代码模式，并将U7/U7-S1下载工具重新上电，如下图所示：（否则将不能正常进行下载）



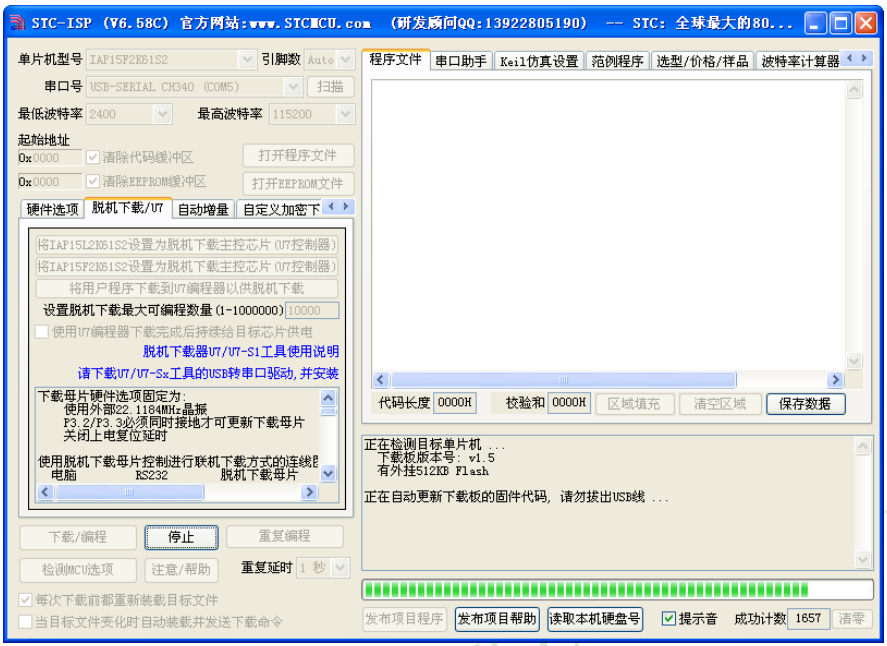
15.6.5.3 自动升级U7/U7-S1

将U7/U7-S1下载板连接到电脑后，打开STC的ISP下载软件，并选择U7/U7-S1所对应的串口。

然后击下载界面中的“下载/编程”、“检测MCU选项”或者“脱机下载/U7”页面里的“将用户程序下载到U7编程器以供脱机下载”按钮中的其中一个时，下载软件便会自动检测U7/U7-S1的版本，若U7/U7-S1的固件版本比当前下载软件里面所记录的U7/U7-S1固件版本低，软件则会弹出如下升级提示：

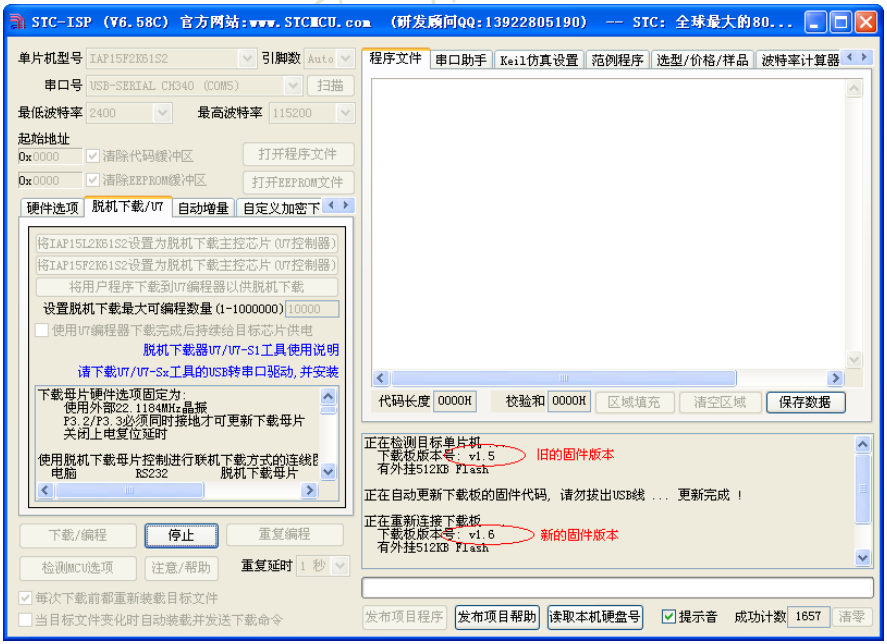


点击“是”，软件便会自动升级U7/U7-S1，如下所示：



自动升级过程中请不要中途拔出USB线。

升级完成后可看到如下的版本变化:



至此，下载软件已自动将U7/U7-S1升级到软件所记录的最新版本（此升级的整个过程中，不需要切换下载板的工作模式，也不需要按更新按钮）

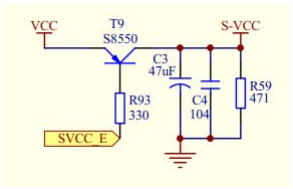
15.6.6 USB型联机/脱机下载板U7-S1的参考电路

USB型联机/脱机下载板 U7 为用户提供了如下的常用控制接口（Ver6.53版）：

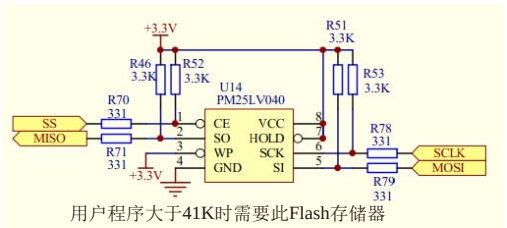
脚位功能	端口	功能描述
电源控制脚	P2.6	低位有效
下载通讯脚	P1.0	串口RXD，连接目标芯片的TXD（P3.1）
	P1.1	串口TXD，连接目标芯片的RXD（P3.0）
编程按键	P3.6	低有效
显示	P3.2	LED1
	P3.3	LED2
	P3.4	LED3
	P5.5	LED4
外挂串行Flash控制脚	P2.4	Flash的CE脚
	P2.2	Flash的SO脚
	P2.3	Flash的SI脚
	P2.1	Flash的SCLK脚
全自动烧录工具 分选机信号	P3.6	起始信号
	P1.5	完成信号
	P5.4	OK信号（良品信号）
	P3.7	ERROR信号（不良品信号）
蜂鸣器（BEEP）控制	P2.5	高有效（高电平发出声音）

STC MCU Limited

电源控制部分参考电路图

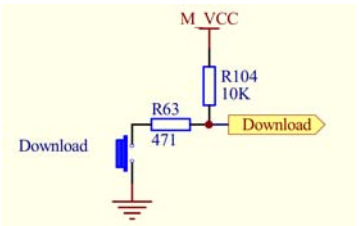


Flash控制部分参考电路图

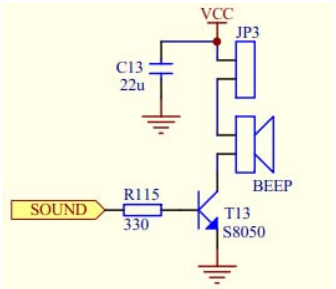


用户程序大于41K时需要此Flash存储器

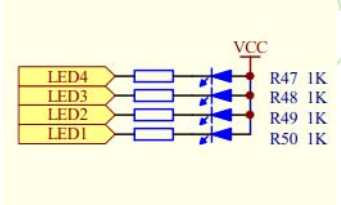
按键部分参考电路图



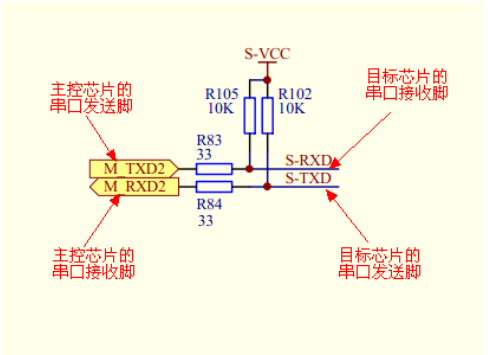
蜂鸣器部分参考电路图



LED显示部分参考电路图

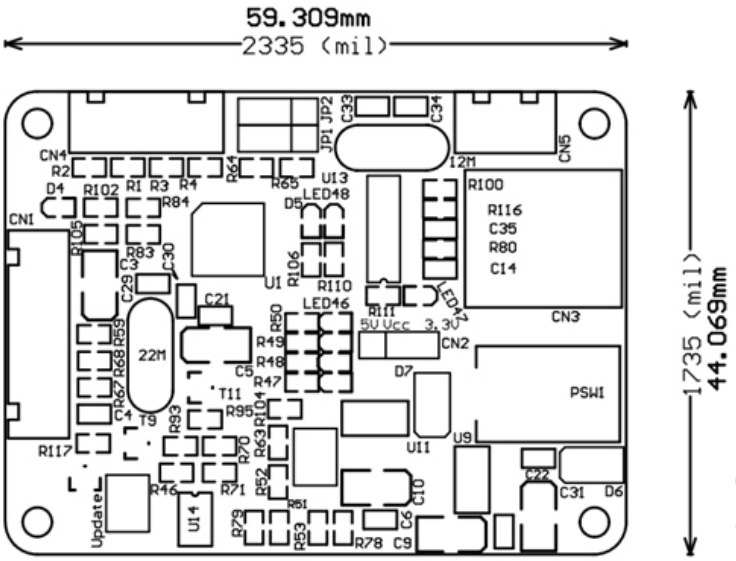


串口通讯脚连接部分参考电路图

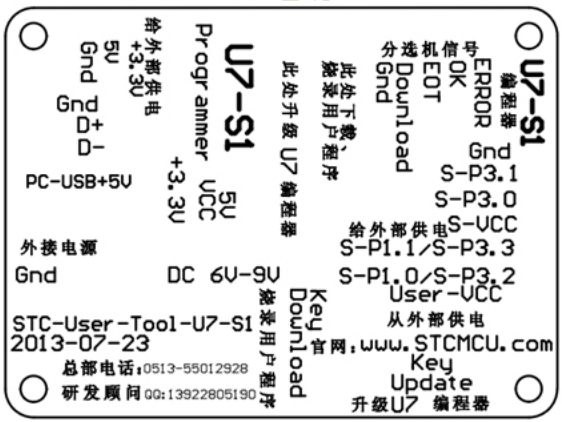


MCU Limited

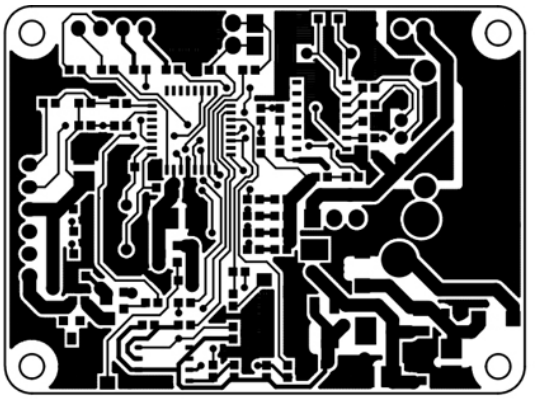
U7-S1 PCB板正面丝印图：



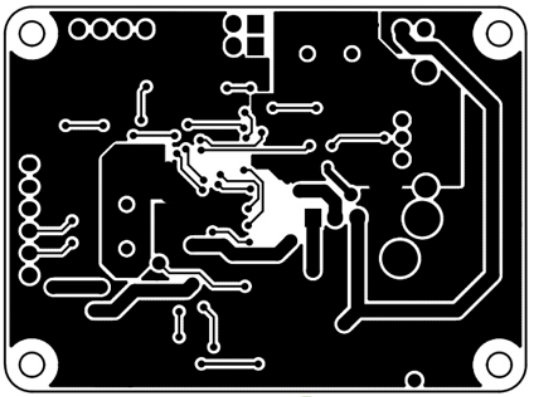
U7-S1 PCB板反面丝印图：



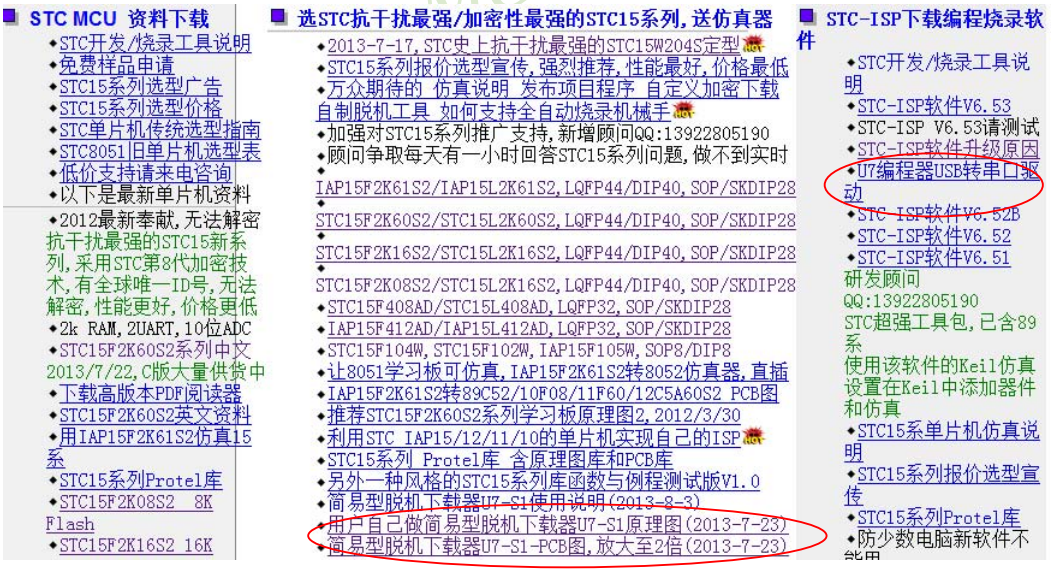
U7-S1 PCB板走线图（正面）：



U7-S1 PCB板走线图（反面）：



在STC官方网站www.STCMCU.com的资源中有用户自己做简易型脱机下载器U7-S1完整原理图以及U7-S1的详细PCB线路图，需要的用户可以自行到网站上下载学习。该原理图及PCB线路图在网站上的位置如下图所示，链接地址分部为：http://www.stcmcu.com/STCISP/用户自己做简易型脱机下载器U7-S1原理图.PDF 及 http://www.stcmcu.com/STCISP/U7-S1-PCB-2x-2013-7-23.PDF 及 U7编程器USB转串口驱动

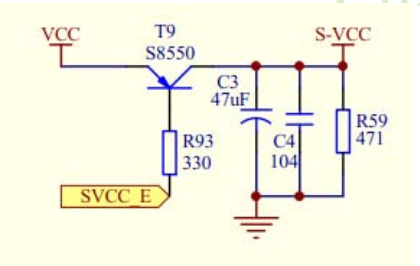


第16章 利用主控芯片对从芯片(限STC15系列)进行ISP下载

STC15系列单片机的用户程序，除可以通过专用的下载工具进行在线联机或离线脱机ISP下载外，还可以利用其他的MCU（如单片机,ARM，DSP等）作主控芯片对其进行ISP下载。

利用其他MCU（如单片机,ARM，DSP等）对STC15系列单片机进行串口ISP下载的系统中，其他MCU为主控芯片，STC15系列单片机为受控的从芯片，主控芯片通过串口控制从芯片进行ISP下载。在利用其他MCU对STC15系列单片机进行ISP下载时，STC15系列单片机（即从芯片）先停电，然后，其他MCU（即主控芯片）发送下载命令给STC15系列单片机（即从芯片），最后，其他MCU控制STC15系列单片机再上电，这样才能正确地对STC15系列单片机进行ISP下载。由于在利用其他MCU（如单片机,ARM，DSP等）对STC15系列单片机进行ISP下载过程中，其他MCU作为主控芯片需要控制从芯片（即STC15系列单片机）电路的电源开关，因此，在进行电路连接时，用户可通过主控芯片的一个I/O口控制从芯片电路的电源开关。STC15系列单片机电路的电源控制部分参考电路图如下：

STC15系列单片机电路的电源控制部分参考电路图



用户可将主控芯片的一个I/O口连接到上图中的SVCC_E管脚，这样通过主控芯片的该I/O口即可控制STC15系列单片机电路的电源开关，那么，在利用其他MCU对STC15系列单片机进行ISP下载时，其他MCU（主控芯片）就可自由控制STC15系列单片机（从芯片）停电或上电了。

利用其他MCU（如单片机,ARM，DSP等）对STC15系列单片机进行串口ISP下载示例程序(C语言程序）如下：

```
/*---------------------------------------------------------------------------------------------------*/
/* --- STC MCU Limited ------------------------------------------------------------------------------*/
/* --- 使用主控芯片对从芯片(限STC15系列)进行ISP下载举例 ---------------------------------------------*/
/* --- Mobile: (86)13922805190 ----------------------------------------------------------------------*/
/* --- Fax: 86-755-82905966 -------------------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 -------------------------------------------------------------------------*/
/* --- Web: www.STCMCU.com --------------------------------------------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了宏晶科技的资料及程序   */
/* 如果要在文章中应用此代码,请在文章中注明使用了宏晶科技的资料及程序   */
/*---------------------------------------------------------------------------------------------------*/
```

```
//本示例在Keil开发环境下请选择Intel的8058芯片型号进行编译
//假定测试芯片的工作频率为11.0592MHz

//注意:使用本代码对STC15系列单片机进行下载时,必须先对STC15系列单片机停电，并要在执行了
// Download代码之后,才给目标芯片上电,否则目标芯片将无法正确下载

#include "reg51.h"

typedef bit BOOL;
typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned long DWORD;

//宏、常量定义
#define   FALSE           0
#define   TRUE            1
#define   LOBYTE(w)       ((BYTE)(WORD)(w))
#define   HIBYTE(w)       ((BYTE)((WORD)(w) >> 8))

#define   MINBAUD         2400L
#define   MAXBAUD         115200L

#define   FOSC            11059200L           //主控芯片工作频率
#define   BR(n)           (65536 - FOSC/4/(n))    //主控芯片串口波特率计算公式
#define   T1MS            (65536 - FOSC/1000)     //主控芯片1ms定时初值

//#define FUSER             11059200L           //STC15系列目标芯片工作频率
//#define FUSER             12000000L           //STC15系列目标芯片工作频率
//#define FUSER             18432000L           //STC15系列目标芯片工作频率
//#define FUSER             22118400L           //STC15系列目标芯片工作频率
#define   FUSER           24000000L           //STC15系列目标芯片工作频率
#define   RL(n)           (65536 - FUSER/4/(n))    //STC15系列目标芯片串口波特率计算公式

//SFR定义
sfr       AUXR    =       0x8e;

//变量定义
BOOL      f1ms;                               //1ms标志位
BOOL      UartBusy;                           //串口发送忙标志位
BOOL      UartReceived;                       //串口数据接收完成标志位
BYTE      UartRecvStep;                       //串口数据接收控制
BYTE      TimeOut;                            //串口通讯超时计数器
BYTE      xdata TxBuffer[256];                //串口数据发送缓冲区
BYTE      xdata RxBuffer[256];                //串口数据接收缓冲区
```

STC MCU Limited

```
char        code DEMO[256];                                    //演示代码数据

//函数声明
void Initial(void);
void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
BOOL Download(BYTE *pdat, long size);

//主函数入口
void main(void)
{
        while (1)
        {
                Initial();

                if (Download(DEMO, 0x0100))
                {
                        //下载成功
                        P3 = 0xff;
                        DelayXms(500);
                        P3 = 0x00;
                        DelayXms(500);
                        P3 = 0xff;
                        DelayXms(500);
                        P3 = 0x00;
                        DelayXms(500);
                        P3 = 0xff;
                        DelayXms(500);
                        P3 = 0x00;
                        DelayXms(500);
                        P3 = 0xff;
                }

                else
                {
                        //下载失败
                        P3 = 0xff;
                        DelayXms(500);
                        P3 = 0xf3;
                        DelayXms(500);
                        P3 = 0xff;
                        DelayXms(500);
```

STC MCU Limited

```
                        P3 = 0xf3;
                        DelayXms(500);
                        P3 = 0xff;
                        DelayXms(500);
                        P3 = 0xf3;
                        DelayXms(500);
                        P3 = 0xff;
                }
        }
}

//1ms定时器中断服务程序
void tm0(void) interrupt 1 using 1
{
        static BYTE Counter100;

        f1ms = TRUE;
        if (Counter100-- == 0)
        {
                Counter100 = 100;
                if (TimeOut) TimeOut--;
        }
}

//串口中断服务程序
void uart(void) interrupt 4 using 1
{
        static WORD RecvSum;
        static BYTE RecvIndex;
        static BYTE RecvCount;
        BYTE dat;

        if (TI)
        {
                TI = 0;
                UartBusy = FALSE;
        }

        if (RI)
        {
                RI = 0;
                dat = SBUF;
                switch (UartRecvStep)
                {
```

STC MCU Limited

```
            case 1:
                if (dat != 0xb9) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 2:
                if (dat != 0x68) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 3:
                if (dat != 0x00) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 4:
                RecvSum = 0x68 + dat;
                RecvCount = dat - 6;
                RecvIndex = 0;
                UartRecvStep++;
                break;
            case 5:
                RecvSum += dat;
                RxBuffer[RecvIndex++] = dat;
                if (RecvIndex == RecvCount) UartRecvStep++;
                break;
            case 6:
                if (dat != HIBYTE(RecvSum)) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 7:
                if (dat != LOBYTE(RecvSum)) goto L_CheckFirst;
                UartRecvStep++;
                break;
            case 8:
                if (dat != 0x16) goto L_CheckFirst;
                UartReceived = TRUE;
                UartRecvStep++;
                break;
            L_CheckFirst:
            case 0:
            default:
                CommInit();
                UartRecvStep = (dat == 0x46 ? 1 : 0);
                break;
        }
    }
```

STC MCU Limited

```
}

//系统初始化
void Initial(void)
{
	UartBusy = FALSE;

	SCON = 0xd0;					//串口数据模式必须为8位数据+1位偶检验
	AUXR = 0xc0;
	TMOD = 0x00;
	TH0 = HIBYTE(T1MS);
	TL0 = LOBYTE(T1MS);
	TR0 = 1;
	TH1 = HIBYTE(BR(MINBAUD));
	TL1 = LOBYTE(BR(MINBAUD));
	TR1 = 1;
	ET0 = 1;
	ES = 1;
	EA = 1;
}

//Xms延时程序
void DelayXms(WORD x)
{
	do
	{
		f1ms = FALSE;
		while (!f1ms);
	} while (x--);
}

//串口数据发送程序
BYTE UartSend(BYTE dat)
{
	while (UartBusy);

	UartBusy = TRUE;
	ACC = dat;
	TB8 = P;
	SBUF = ACC;

	return dat;
}
```

STC MCU Limited

```
//串口通讯初始化
void CommInit(void)
{
	UartRecvStep = 0;
	TimeOut = 20;
	UartReceived = FALSE;
}

//发送串口通讯数据包
void CommSend(BYTE size)
{
	WORD sum;
	BYTE i;

	UartSend(0x46);
	UartSend(0xb9);
	UartSend(0x6a);
	UartSend(0x00);
	sum = size + 6 + 0x6a;
	UartSend(size + 6);
	for (i=0; i<size; i++)
	{
		sum += UartSend(TxBuffer[i]);
	}
	UartSend(HIBYTE(sum));
	 UartSend(LOBYTE(sum));
	UartSend(0x16);
	while (UartBusy);

	CommInit();
}

//对STC15系列的芯片进行数据下载程序
BOOL Download(BYTE *pdat, long size)
{
	BYTE arg;
	BYTE cnt;
	WORD addr;

	//握手
	CommInit();
	while (1)
	{
		if (UartRecvStep == 0)
```

STC MCU Limited

```
        {
            UartSend(0x7f);
            DelayXms(10);
        }
        if (UartReceived)
        {
            arg = RxBuffer[4];
            if (RxBuffer[0] == 0x50) break;
            return FALSE;
        }
    }

    //设置参数
    TxBuffer[0] = 0x01;
    TxBuffer[1] = arg;
    TxBuffer[2] = 0x40;
    TxBuffer[3] = HIBYTE(RL(MAXBAUD));
    TxBuffer[4] = LOBYTE(RL(MAXBAUD));
    TxBuffer[5] = 0x00;
    TxBuffer[6] = 0x00;
    TxBuffer[7] = 0xc3;
    CommSend(8);
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x01) break;
            return FALSE;
        }
    }

     //准备
    TH1 = HIBYTE(BR(MAXBAUD));
    TL1 = LOBYTE(BR(MAXBAUD));
    DelayXms(10);
    TxBuffer[0] = 0x05;
    CommSend(1);
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x05) break;
```

STC MCU Limited

```
                return FALSE;
        }
}
 //擦除
DelayXms(10);
TxBuffer[0] = 0x03;
TxBuffer[1] = 0x00;
CommSend(2);
TimeOut = 100;
while (1)
{
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
                if (RxBuffer[0] == 0x03) break;
                return FALSE;
        }
}
//写用户代码
DelayXms(10);
addr = 0;
TxBuffer[0] = 0x22;
while (addr < size)
{
        TxBuffer[1] = HIBYTE(addr);
        TxBuffer[2] = LOBYTE(addr);
        cnt = 0;
        while (addr < size)
        {
                TxBuffer[cnt+3] = pdat[addr];
                addr++;
                cnt++;
                if (cnt >= 128) break;
        }
        CommSend(cnt + 3);
        while (1)
        {
                if (TimeOut == 0) return FALSE;
                if (UartReceived)
                {
                        if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
                        return FALSE;
                }
        }
```

STC MCU Limited

```
                TxBuffer[0] = 0x02;
        }
        //写硬件选项(如果不需要修改硬件选项,此步骤可直接跳过)
        //说明：  在利用其他MCU（如单片机,ARM，DSP等）对STC15系列单片机进行串口ISP下载时，
        //        STC15系列单片机的工作时钟频率默认为24MHz，此步骤中的时钟设置即是将STC15系列
        //        单片机的工作时钟设置为24MHz时钟。
        //        如果用户不需要修改硬件选项,此步骤可直接跳过,此时所有的硬件选项都维持不变,
        //        STC15系列单片机的频率为上一次所调节频率，反之若写改硬件选项,则STC15系列单片
        //        机的内部IRC频率将被固定写为24M。
        //建议：  第一次使用STC-ISP下载软件将从芯片的硬件选项设置好，
        //        以后再使用主控芯片对从芯片下载程序时不写硬件选项
        DelayXms(10);
        for (cnt=0; cnt<128; cnt++)
        {
                TxBuffer[cnt] = 0xff;
        }
        TxBuffer[0] = 0x04;
        TxBuffer[1] = 0x00;
        TxBuffer[2] = 0x00;
        TxBuffer[34] = 0xfd;
        TxBuffer[62] = arg;
        TxBuffer[63] = 0x7f;
        TxBuffer[64] = 0xf7;
        TxBuffer[65] = 0x7b;
        TxBuffer[66] = 0x1f;
        CommSend(67);
        while (1)
        {
                if (TimeOut == 0) return FALSE;
                if (UartReceived)
                {
                        if ((RxBuffer[0] == 0x04) && (RxBuffer[1] == 'T')) break;
                        return FALSE;
                }
        }

        //下载完成
         return TRUE;
}

char code DEMO[256] =
{
        0x02,   0x00,   0x5E,   0x12,   0x00,   0x4B,   0x75,   0xB0,
        0xEF,   0x12,   0x00,   0x2C,   0x75,   0xB0,   0xDF,   0x12,
```

STC MCU Limited

```
        0x00,   0x2C,   0x75,   0xB0,   0xFE,   0x12,   0x00,   0x2C,
        0x75,   0xB0,   0xFD,   0x12,   0x00,   0x2C,   0x75,   0xB0,
        0xFB,   0x12,   0x00,   0x2C,   0x75,   0xB0,   0xF7,   0x12,
        0x00,   0x2C,   0x80,   0xDA,   0xE4,   0xFF,   0xFE,   0xE4,
        0xFD,   0xFC,   0x0D,   0xBD,   0x00,   0x01,   0x0C,   0xBC,
        0x01,   0xF8,   0xBD,   0xF4,   0xF5,   0x0F,   0xBF,   0x00,
        0x01,   0x0E,   0xBE,   0x03,   0xEA,   0xBF,   0xE8,   0xE7,
        0x02,   0x00,   0x4B,   0x75,   0x80,   0xFF,   0x75,   0x90,
        0xFF,   0x75,   0xA0,   0xFF,   0x75,   0xB0,   0xFF,   0x75,
        0xC0,   0xFF,   0x75,   0xC8,   0xFF,   0x22,   0x78,   0x7F,
        0xE4,   0xF6,   0xD8,   0xFD,   0x75,   0x81,   0x07,   0x02,
        0x00,   0x03,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
        0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,   0x00,
};
```

STC MCU Limited

附录A：STC15W104SW系列单片机总体介绍

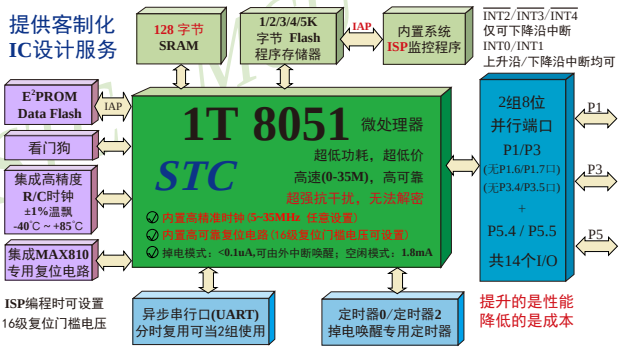
A.1 STC15W104SW系列简介(用STC15W201S系列取代)

—— 管脚P1.0不要用，INT0/INT1/$\overline{INT2}$/$\overline{INT3}$/$\overline{INT4}$不要用

STC15W104SW系列单片机是STC生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/超强抗干扰的新一代8051单片机，采用STC第九代加密技术，超级加密，指令代码完全兼容传统8051,但速度快8-12倍。内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP编程时16级复位门槛电压可选)。1组高速异步串行通信口(UART)，可在两组管脚之间进行切换，分时复用可作两组串行口使用，针对串行口通信/电机控制/强干扰场合。

在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.



1. 增强型 8051 CPU，1T，单时钟/机器周期，速度比普通8051快8-12倍
2. 工作电压：5.5V - 2.4V
3. 2K/4K/5K字节片内Flash程序存储器，擦写次数10万次以上
4. 片内128字节的SRAM
5. 有片内EEPROM功能，擦写次数10万次以上
6. ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
7. 内部高可靠复位，ISP编程时16级复位门槛电压可选，可彻底省掉外部复位电路
8. 内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，内部时钟从5MHz~35MHz可选(5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz)
9. 工作频率范围：5MHz ~ 35MHz，相当于普通8051的60MHz～420MHz

10.一组高速异步行串行口，可在两组管脚之间进行切换，分时复用可当两组串口使用：
串口(RxD/P3.0, TxD/P3.1)可以切换到(RxD_2/P3.6, TxD_2/P3.7).

11.低功耗设计：低速模式，空闲模式，掉电模式/停机模式.

12.可将掉电模式/停机模式唤醒的定时器：有内部低功耗掉电唤醒专用定时器。

13.可将掉电模式/停机模式唤醒的资源有：INT0/P3.2, INT1/P3.3 (INT0/INT1上升沿下降沿中断均可), $\overline{\text{INT2}}$/P3.6, $\overline{\text{INT3}}$/P3.7, $\overline{\text{INT4}}$/P3.0($\overline{\text{INT2}}$/$\overline{\text{INT3}}$/$\overline{\text{INT4}}$仅可下降沿中断)；管脚RxD(可在RxD/P3.0和RxD_2/P3.6之间切换)；管脚T0/T2(下降沿，不产生中断，前提是在进入掉电模式/停机模式前定时器中断已经被允许)；内部掉电唤醒专用定时器。

14. 共2个定时器/计数器——T0(兼容普通8051的定时器)/T2。定时器/计数器0和定时器/计数器2都是16位可重装载的定时器/计数器，既可作定时器(对内部系统时钟进行计数)也可作计数器器(对外部管脚的时钟输入进行计数)，并可实现可编程时钟输出T0CLKO/T2CLKO，还可对外部管脚的时钟输入进行时钟分频输出作分频器使用。

15. 可编程时钟输出功能(对内部系统时钟或对外部管脚的时钟输入进行时钟分频输出)：
由于STC15系列5V单片机I/O口的对外输出速度最快不超过13.5MHz，所以5V单片机的对外可编程时钟输出速度最快也不超过13.5MHz.；
而3.3V单片机I/O口的对外输出速度最快不超过8MHz，故3.3V单片机的对外可编程时钟输出速度最快也不超过8MHz.
① T0在P1.4/T0CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T0/P1.2的时钟输入进行可编程时钟分频输出)；
② T2在P3.0/T2CLKO进行可编程输出时钟(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出)；
③ 以上2个定时器/计数器均可1~65536级分频输出。
主时钟在P5.4/MCLKO对外输出时钟，并可如下分频MCLK/1, MCLK/2, MCLK/4.
主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。MCLK是指主时钟频率，MCLKO是指主时钟输出。
STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机、均在MCLKO/P5.4口对外输出主时钟。

16. 硬件看门狗(WDT)

17. 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令

18. 通用I/O口（14个），复位后为：准双向口/弱上拉（普通8051传统I/O口）
可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
每个I/O口驱动能力均可达到20mA，但整个芯片电流最大不要超过90mA.
如果I/O口不够用，可外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口。

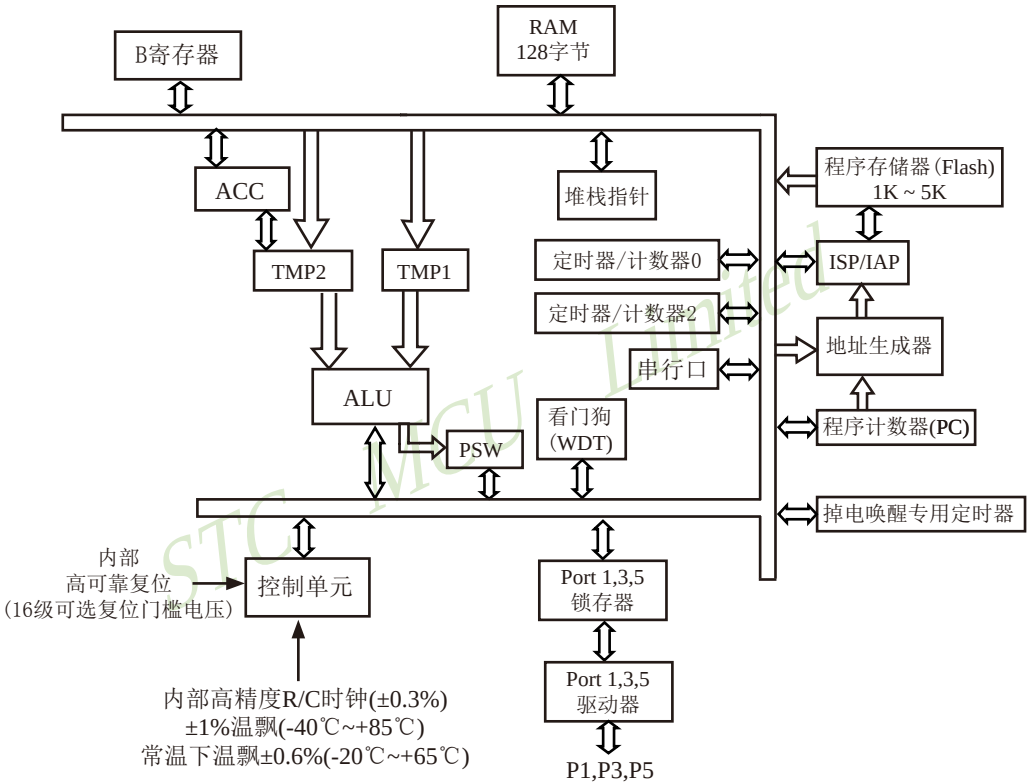
19. 封装：SOP16, DIP16.

20. 全部175℃八小时高温烘烤，高品质制造保证

21. 开发环境：在 Keil C 开发环境中，选择 Intel 8052 编译，头文件包含<reg51.h>即可

A.2 STC15W104SW系列单片机的内部结构图

STC15W104SW系列单片机的内部结构框图如下图所示。STC15W104SW系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器/计数器、掉电唤醒专用定时器、I/O口、看门狗、高速异步串行通信端口UART、片内高精度R/C时钟及高可靠复位等模块。



STC15W104SW系列内部结构框图

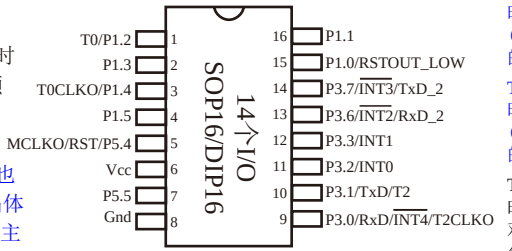
A.3 STC15W104SW系列单片机管脚图

所有封装形式均满足欧盟RoHS要求，强烈推荐选择SOP16贴片封装，传统的插件DIP16封装稳定供货。

中国大陆本土STC姚永平独立创新设计：
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

MCLKO是指主时钟输出，主时钟对外输出的时钟可如下分频
MCLK/1, MCLK/2, MCLK/4.

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。MCLK是指主时钟频率。



T0CLKO是指定时器/计数器0的可编程时钟输出
(对内部系统时钟或对外部管脚T0/P1.2的时钟输入进行可编程时钟分频输出);

T2CLKO是指定时器/计数器2的可编程时钟输出
(对内部系统时钟或对外部管脚T2/P3.1的时钟输入进行可编程时钟分频输出);

T0CLKO/T2CLKO除可以对内部系统时钟进行可编程时钟输出外，还可以对外部管脚T0/T2的时钟输入进行时钟分频输出，作分频器使用。

对于STC15系列5V单片机，由于I/O口的对外输出速度最快不超过13.5MHz，所以对外可编程时钟输出速度最快也不超过13.5MHz;
对于3.3V单片机，由于I/O口的对外输出速度最快不超过8MHz，所以对外可编程时钟输出速度最快也不超过8MHz;

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	x1xx,xx0x
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	00x0,x000

串口1/S1可在2个地方切换，由 S1_S0 控制位来选择		
S1_S1	S1_S0	串口1/S1可在P1/P3之间来回切换
0	0	串口1/S1在[P3.0/RxD, P3.1/TxD]
0	1	串口1/S1在[P3.6/RxD_2, P3.7/TxD_2]
1	0	串口1/S1在[P1.6/RxD_3/XTAL2, P1.7/TxD_3/XTAL1] 串口1在P1口时要使用内部时钟
1	1	无效

MCKO_S1	MCKO_S0	主时钟对外分频输出控制位 (主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟)
0	0	主时钟不对外输出时钟
0	1	主时钟对外输出时钟，但时钟频率不被分频，输出时钟频率 = MCLK / 1
1	0	主时钟对外输出时钟，但时钟频率被2分频，输出时钟频率 = MCLK / 2
1	1	主时钟对外输出时钟，但时钟频率被4分频，输出时钟频率 = MCLK / 4

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK是指主时钟频率。
STC15W104SW系列单片机在MCLKO/P5.4口对外输出主时钟。
STC15系列8-pin单片机(如STC15F101W系列)在MCLKO/P3.4口对外输出主时钟，STC15系列16-pin及其以上单片机(如STC15W4K60S4系列和STC15F2K60S2系列)均在MCLKO/P5.4口对外输出主时钟。

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0	00x0,x000

Tx_Rx：串口1的中继广播方式设置

0：串口1为正常工作方式

1：串口1为中继广播方式，即将RxD端口输入的电平状态实时输出在TxD外部管脚上，TxD外部管脚可以对RxD管脚的输入信号进行实时整形放大输出，TxD管脚的对外输出实时反映RxD端口输入的电平状态。

串口1的RxD管脚和TxD管脚可以在2组不同管脚之间进行切换：[RxD/P3.0, TxD/P3.1]；
[RxD_2/P3.6, TxD_2/P3.7].

CLKS2	CLKS1	CLKS0	系统时钟选择控制位 （系统时钟是指对主时钟进行分频后供给CPU、串行口、定时器的实际工作时钟）
0	0	0	主时钟频率/1, 不分频
0	0	1	主时钟频率/2
0	1	0	主时钟频率/4
0	1	1	主时钟频率/8
1	0	0	主时钟频率/16
1	0	1	主时钟频率/32
1	1	0	主时钟频率/64
1	1	1	主时钟频率/128

主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟。

STC MCU Limited

A.4 STC15W104SW系列单片机选型价格一览表

型号	工作电压（V）	Flash程序存储器(byte)	大容量SRAM字节	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	内部复位(可选复位门槛电压)	看门狗	所有封装SOP16/DIP16价格(RMB ¥)	
																	SOP16	DIP16
STC15W104SW系列单片机选型价格一览表																		
STC15W102SW	5.5-2.4	2K	128	1	–	2	-	有	5	–	–	1	3K	有	16级	有		
STC15W104SW	5.5-2.4	4K	128	1	–	2	-	有	5	–	–	1	1K	有	16级	有	¥1.7	¥1.9
IAP15W105SW	5.5-2.4	5K	128	1	–	2	-	有	5	–	–	1	IAP	有	16级	有	¥1.7 用户可将用户程序区的程序Flash当EEPROM使用	¥1.9

提供客制化IC服务

建议用户选用SOP16封装，但DIP16封装仍正常供货

封装为DIP16的单片机比封装为SOP16的单片机要贵0.2元；

以上单价为200K起订

量小每片需加0.1元

以上价格运费由客户承担,零售10片起

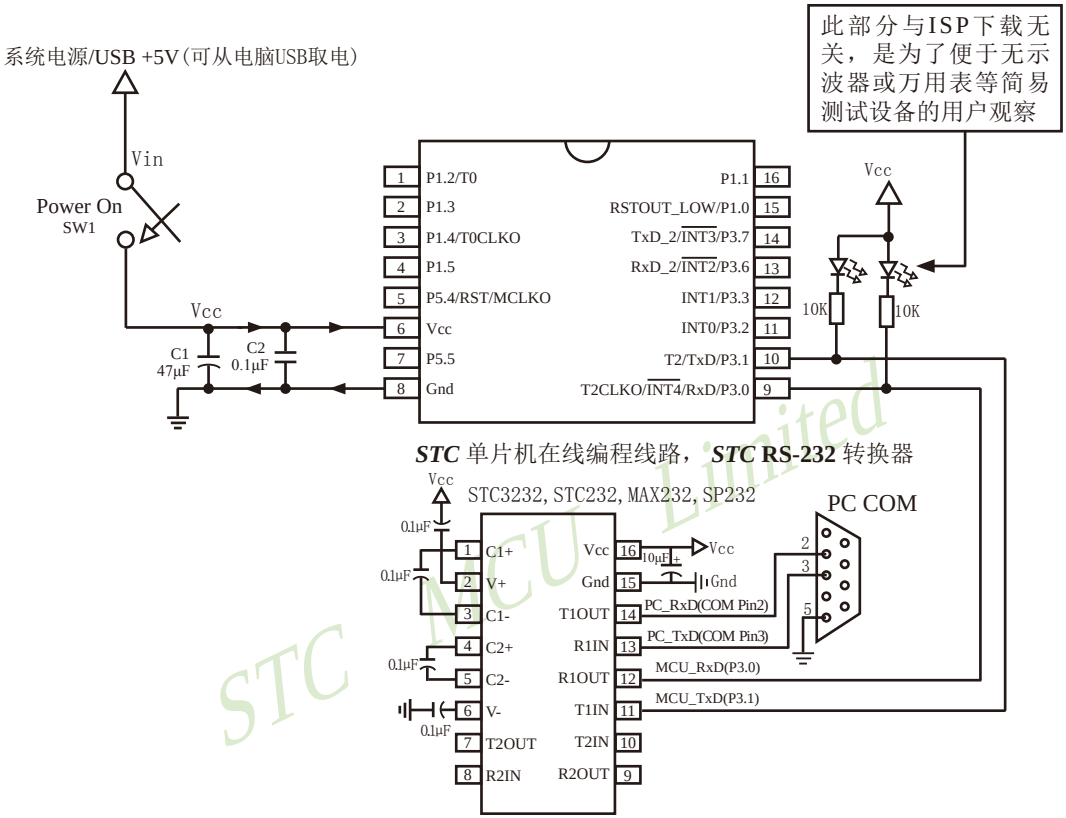
如对价格不满，可来电要求降价

总结：STC15W104SW系列单片机有：两个16位重装载定时器/计数器(这两个定时器/计数器分别是：定时器/计数器0和定时器/计数器2)；5个支持掉电唤醒的外部中断INT0/INT1/INT2/INT3/INT4；1组高速异步串行通信端口；1个数据指针DPTR等功能。表中“–”表示该型号的单片机无相应的功能。STC15W104SW系列单片机无SPI、无A/D转换、无CCP/PWM/PCA、无掉电唤醒定时器、无外部数据总线等功能。

因为程序区的最后7个字节单元被强制性的放入全球唯一ID号的内容，所以用户实际可以使用的程序空间大小要比选型表中的大小少7个字节。

STC MCU Limited

A.5 STC15W104SW单片机在系统可编程典型应用(ISP)线路图



内部高可靠复位，可彻底省掉外部复位电路

P5.4/RST/MCLKO脚出厂时默认为I/O口，可以通过 STC-ISP 编程器将其设置为RST复位脚(高电平复位).

内部集成高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)，5MHz~35MHz宽范围可设置，可彻底省掉外部昂贵的晶振

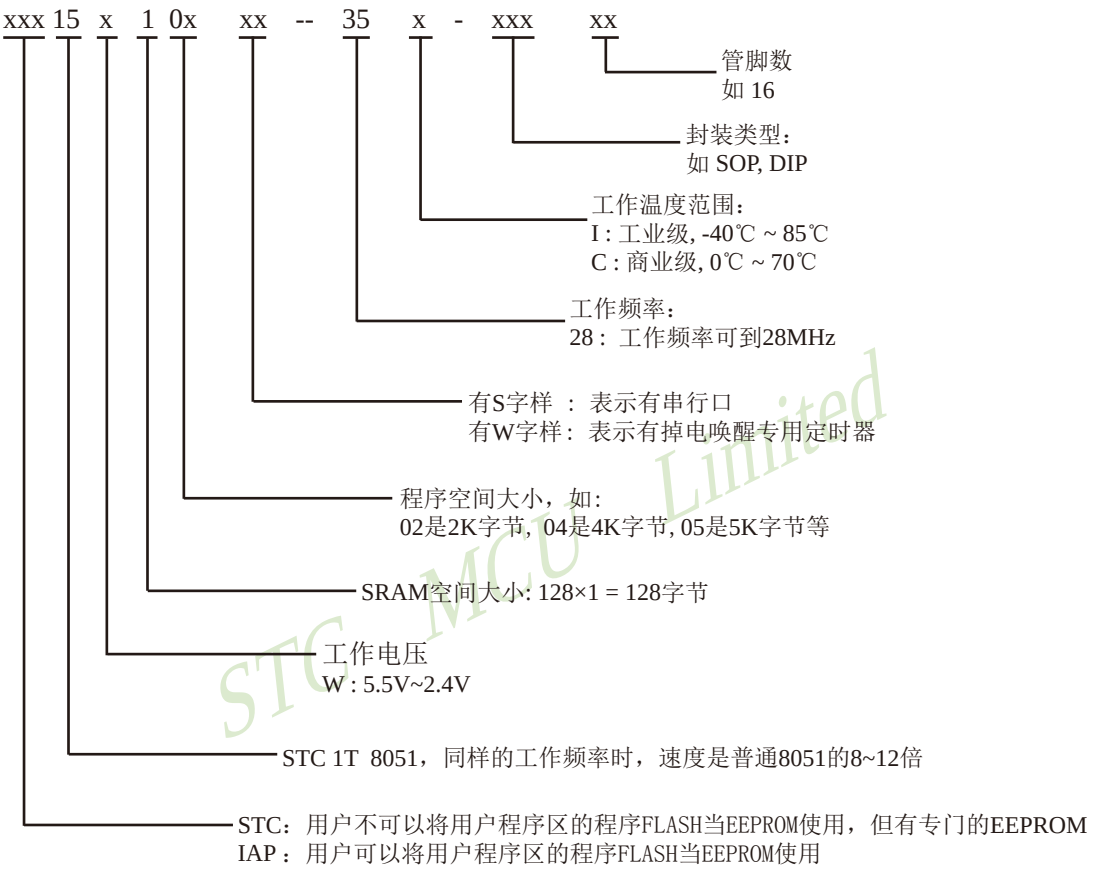
建议在Vcc和Gnd之间就近加上电源去耦电容C1(47μF), C2(0.1μF), 可去除电源线噪声，提高抗干扰能力

A.6 STC15W104SW系列单片机的管脚说明

管脚	管脚编号(封装SOP16/DIP16)	说明	
P1.0/RSTOUT_LOW	15	P1.0	标准I/O口 PORT1[0]
		RSTOUT_LOW	上电后,输出低电平，在复位期间也是输出低电平，用户可用软件将其设置为高电平或低电平，如果要读外部状态，可将该口先置高后再读
P1.1	16	标准I/O口 PORT1[1]	
P1.2/T0	1	P1.2	标准I/O口 PORT1[2]
		T0	定时器/计数器0的外部输入
P1.3	2	标准I/O口 PORT1[3]	
P1.4/T0CLKO	3	P1.4	标准I/O口 PORT1[4]
		T0CLKO	T0的时钟输出 可通过设置INT_CLKO[0]位/T0CLKO将该管脚配置为T0CLKO, 也可对T0脚的外部时钟输入进行分频输出
P1.5	4	标准I/O口 PORT1[5]	
P3.0/RxD/$\overline{\text{INT4}}$/T2CLKO	9	P3.0	标准I/O口 PORT3[0]
		RxD	串口数据接收端
		$\overline{\text{INT4}}$	外部中断4,只能下降沿中断, $\overline{\text{INT4}}$支持掉电唤醒
		T2CLKO	T2的时钟输出 可通过设置INT_CLKO[2]位/T2CLKO将该管脚配置为T2CLKO
P3.1/TxD/T2	10	P3.1	标准I/O口 PORT3[1]
		TxD	串口数据发送端
		T2	定时器/计数器2的外部输入
P3.2/INT0	11	P3.2	标准I/O口 PORT3[2]
		INT0	外部中断0，既可上升沿中断也可下降沿中断. 如果IT0(TCON.0)被置为1, INT0管脚仅为下降沿中断。如果IT0(TCON.0)被清0，INT0管脚既支持上升沿中断也支持下降沿中断。 INT0支持掉电唤醒。
P3.3/INT1	12	P3.3	标准I/O口 PORT3[3]
		INT1	外部中断1，既可上升沿中断也可下降沿中断. 如果IT1(TCON.2)被置为1, INT1管脚仅为下降沿中断。如果IT1(TCON.2)被清0，INT1管脚既支持上升沿中断也支持下降沿中断。 INT1支持掉电唤醒。
P3.6/$\overline{\text{INT2}}$/RxD_2	13	P3.6	标准I/O口 PORT3[6]
		$\overline{\text{INT2}}$	外部中断2,只能下降沿中断 $\overline{\text{INT2}}$支持掉电唤醒
		RxD_2	串口数据接收端
P3.7/$\overline{\text{INT3}}$/TxD_2	14	P3.7	标准I/O口 PORT3[7]
		$\overline{\text{INT3}}$	外部中断3,只能下降沿中断 $\overline{\text{INT3}}$支持掉电唤醒
		TxD_2	串口数据发送端
P5.4/RST/MCLKO	5	P5.4	标准I/O口 PORT5[4]
		RST	复位脚(高电平复位)
		MCLKO	主时钟输出;输出的频率可为MCLK/1, MCLK/2, MCLK/4。 主时钟可以是内部R/C时钟，也可是外部输入的时钟或外部晶体振荡产生的时钟，MCLK指主时钟频率。
P5.5	7	标准I/O口 PORT5[5]	
Vcc	6	电源正极	
Gnd	8	电源负极，接地	

STC MCU Limited

A.7 STC15W104SW系列单片机命名规则



STC MCU Limited

附录B：STC15系列单片机电气特性

Absolute Maximum Ratings

Parameter	Symbol	Min	Max	Unit
Srotage temperature	TST	-55	+125	℃
Operating temperature (I)	TA	-40	+85	℃
Operating temperature (C)	TA	0	+70	℃
DC power supply (5V)	VDD - VSS	-0.3	+5.5	V
DC power supply (3V)	VDD - VSS	-0.3	+3.6	V
Voltage on any pin	-	-0.3	VCC + 0.3	V

DC Specification (5V MCU)

Sym	Parameter	Specification				Test Condition
		Min.	Typ	Max.	Unit	
VDD	Operating Voltage	3.3	5.0	5.5	V	
IPD	Power Down Current	-	< 0.1	-	uA	5V
IIDL	Idle Current	-	3.0	-	mA	5V
ICC	Operating Current	-	4	20	mA	5V
VIL1	Input Low (P0,P1,P2,P3)	-	-	0.8	V	5V
VIH1	Input High (P0,P1,P2,P3)	2.0	-	-	V	5V
VIH2	Input High (RESET)	2.2	-	-	V	5V
IOL1	Sink Current for output low (P0,P1,P2,P3)	-	20	-	mA	5V@Vpin=0.45V
IOH1	Sourcing Current for output high (P0,P1,P2,P3) (Quasi-output)	200	270	-	uA	5V
IOH2	Sourcing Current for output high (P0,P1,P2,P3) (Push-Pull, Strong-output)	-	20	-	mA	5V@Vpin=2.4V
IIL	Logic 0 input current (P0,P1,P2,P3)	-	-	50	uA	Vpin=0V
ITL	Logic 1 to 0 transition current (P0,P1,P2,P3)	100	270	600	uA	Vpin=2.0V

DC Specification (3V MCU)

Sym	Parameter	Specification				Test Condition
		Min.	Typ	Max.	Unit	
VDD	Operating Voltage	2.4	3.3	3.6	V	
IPD	Power Down Current	-	<0.1	-	uA	3.3V
IIDL	Idle Current	-	2.0	-	mA	3.3V
ICC	Operating Current	-	4	10	mA	3.3V
VIL1	Input Low (P0,P1,P2,P3)	-	-	0.8	V	3.3V
VIH1	Input High (P0,P1,P2,P3)	2.0	-	-	V	3.3V
VIH2	Input High (RESET)	2.2	-	-	V	3.3V
IOL1	Sink Current for output low (P0,P1,P2,P3)	-	20	-	mA	3.3V@Vpin=0.45V
IOH1	Sourcing Current for output high (P0,P1,P2,P3) (Quasi-output)	140	170	-	uA	3.3V
IOH2	Sourcing Current for output high (P0,P1,P2,P3) (Push-Pull)	-	20	-	mA	3.3V
IIL	Logic 0 input current (P0,P1,P2,P3)	-	8	50	uA	Vpin=0V
ITL	Logic 1 to 0 transition current (P0,P1,P2,P3)	-	110	600	uA	Vpin=2.0V

STC MCU Limited

附录C：内部常规256字节RAM间接寻址测试程序

```
;/* ---  STC International Limited ------------------------------------------ */
;/* ---  STC     姚永平     2006/1/6     V1.0 ------------------------------------ */
;/* ---  STC15 系列单片机 内部常规RAM间接寻址测试程序-------------------------- */
;/* ---  研发顾问QQ: 13922805190 -------------------------------------------- */
;/* ---  Fax: 0755-82905966 ------------------------------------------------- */
;/* ---  Tel: 0755-82948409 ------------------------------------------------- */
;/* ---  Web: www.STCMCU.com ------------------------------------------------ */
;/* ---  本演示程序在STC 15系列 ISP的下载编程工具上测试通过 ------------------- */
;/* ---  如果要在程序中使用该程序,请在程序中注明使用了STC的资料及程序  -------- */
;/* ---  如果要在文章中引用该程序,请在文章中注明使用了STC的资料及程序  -------- */

TEST_CONST   EQU   5AH
;TEST_RAM    EQU   03H
        ORG   0000H
        LJMP  INITIAL

        ORG   0050H
INITIAL:
        MOV   R0,   #253

        MOV   R1,   #3H
TEST_ALL_RAM:
        MOV   R2,   #0FFH
TEST_ONE_RAM:
        MOV   A,    R2
        MOV   @R1,  A
        CLR   A
        MOV   A,    @R1

        CJNE  A,    2H,     ERROR_DISPLAY
        DJNZ  R2,   TEST_ONE_RAM
        INC   R1
        DJNZ  R0,   TEST_ALL_RAM
```

STC MCU Limited

```
OK_DISPLAY:
        MOV     P1,     #11111110B
Wait1:
        SJMP    Wait1

ERROR_DISPLAY:
        MOV     A,      R1
        MOV     P1,     A
Wait2:
        SJMP    Wait2
        END
```

STC MCU Limited

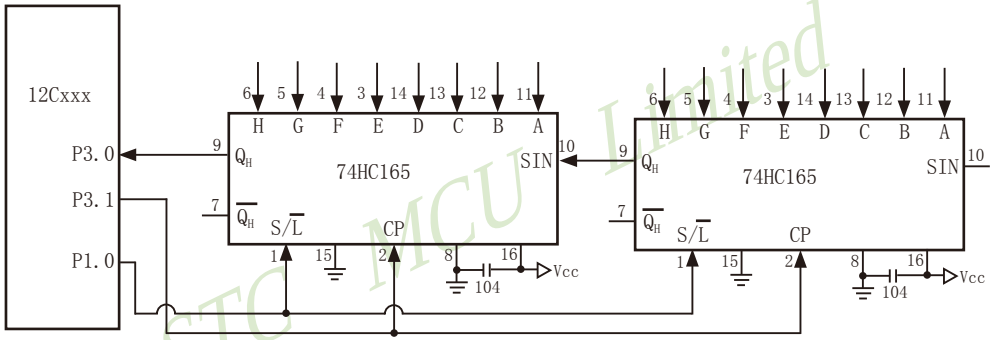
附录D：用串口扩展I/O接口

STC15系列单片机串行口的方式0可用于I/O扩展。如果在应用系统中，串行口未被占用，那么将它用来扩展并行I/O口是一种经济、实用的方法。

在操作方式0时，串行口作同步移位寄存器，其波特率是固定的，为SYSclk/12（SYSclk为系统时钟频率）。数据由RXD端（P3.0）出入，同步移位时钟由TXD端（P3.1）输出。发送、接收的是8位数据，低位在先。

一、用74HC165扩展并行输入口

下图是利用两片74HC165扩展二个8位并行输入口的接口电路图。



74HC165是8位并行置入移位寄存器。当移位/置入端(S/L)由高到低跳变时，并行输入端的数据置入寄存器；当S/L=1，且时钟禁止端（第15脚）为低电平时，允许时钟输入，这时在时钟脉冲的作用下，数据将由Q_A到Q_H方向移位。

上图中，TXD(P3.1)作为移位脉冲输出端与所有74HC165的移位脉冲输入端CP相连；RXD（P3.0）作为串行输入端与74HC165的串行输出端Q_H相连；P1.0用来控制74HC165的移位与置入而同S/L相连；74HC165的时钟禁止端（15脚）接地，表示允许时钟输入。当扩展多个8位输入口时，两芯片的首尾（Q_H与S_{IN}）相连。

下面的程序是从16位扩展口读入5组数据（每组二个字节），并把它们转存到内部RAM 20H开始的单元中。

STC MCU Limited

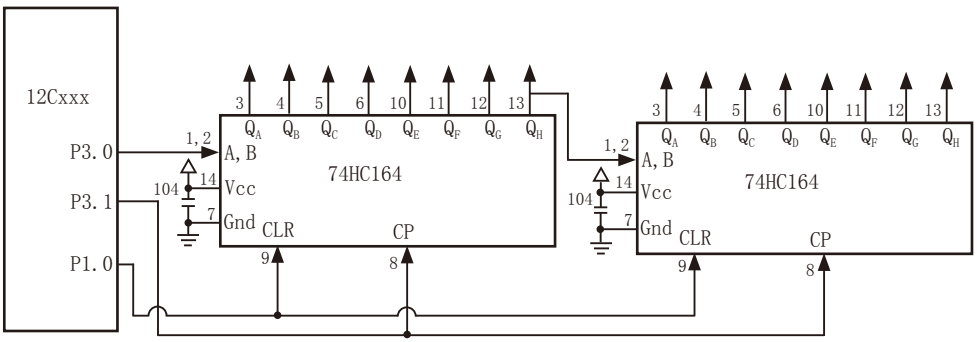
```
                MOV     R7,     #05H            ；设置读入组数
                MOV     R0,     #20H            ；设置内部RAM数据区首址
START:          CLR     P1.0                    ；并行置入数据，S/L=0
                SETB    P1.0                    ；允许串行移位S/L=1
                MOV     R1,     #02H        ；设置每组字节数，即外扩74LS165的个数
RXDATA:         MOV     SCON,   #00010000B  ；设串行方式0，允许接收，启动接收过程
WAIT:           JNB     RI,     WAIT            ；未接收完一帧，循环等待
                CLR     RI                      ；清RI标志，准备下次接收
                MOV     A,      SBUF            ；读入数据
                MOV     @R0,    A               ；送至RAM缓冲区
                INC     R0                      ；指向下一个地址
                DJNZ    R1,     RXDATA          ；为读完一组数据，继续
                DJNZ    R7,     START           ；5组数据未读完重新并行置入
                ……                              ；对数据进行处理
```

STC MCU Limited

上面的程序对串行接收过程采用的是查询等待的控制方式，如有必要，也可改用中断方式。从理论上讲，按上图方法扩展的输入口几乎是无限的，但扩展的越多，口的操作速度也就越慢。

二、用74HC164扩展并行输出口

74HC164是8位串入并出移位寄存器。下图是利用74HC164扩展二个8位输出口的接口电路。



当单片机串行口工作在方式0的发送状态时，串行数据由P3.0（RXD）送出，移位时钟由P3.1（TXD）送出。在移位时钟的作用下，串行口发送缓冲器的数据一位一位地移入74HC164中。需要指出的是，由于74HC164无并行输出控制端，因而在串行输入过程中，其输出端的状态会不断变化，故在某些应用场合，在74HC164的输出端应加接输出三态门控制，以便保证串行输入结束后再输出数据。

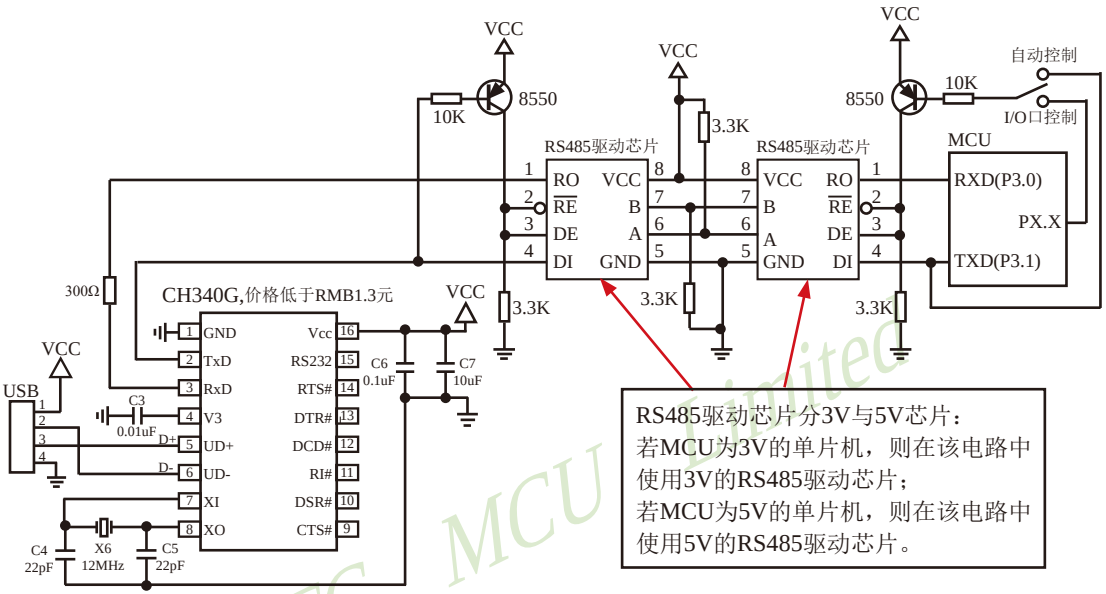
下面是将RAM缓冲区30H、31H的内容串行口由74HC164并行输出的子程序。

```
START：
        MOV         R7,     #02H        ；设置要发送的字节个数
        MOV         R0,     #30H        ；设置地址指针
        MOV         SCON,   #00H        ；设置串行口方式0
SEND：
        MOV         A,      @R0
        MOV         SBUF,   A           ；启动串行口发送过程
WAIT：
        JNB         TI,     WAIT        ；一帧数据未发送完，循等待
        CLR         TI
        INC         R0                  ；取下一个数
        DJNZ        R7,     SEND
        RET
```

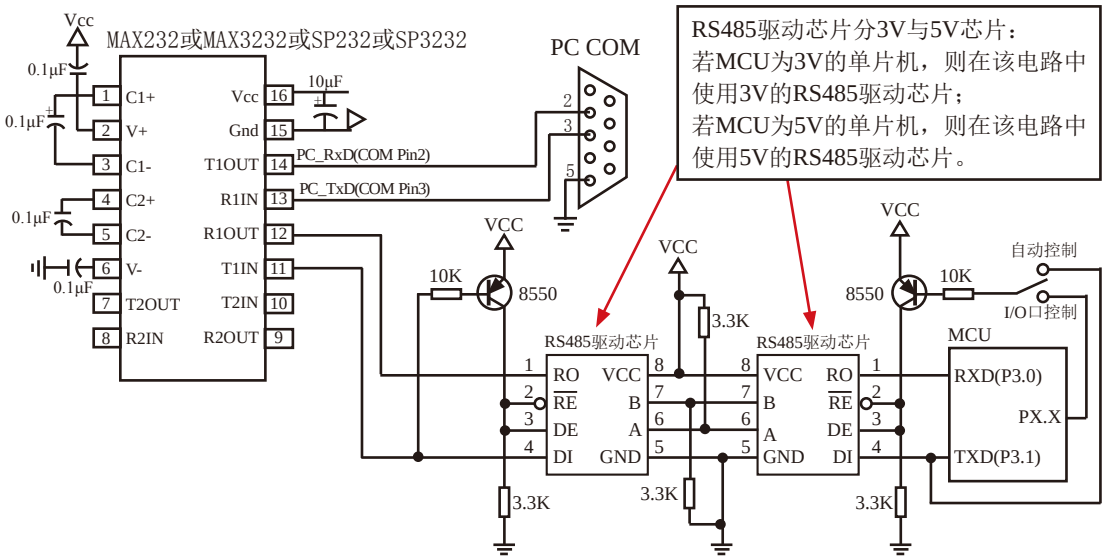
STC MCU Limited

附录E：RS485自动控制或I/O口控制下载线路图

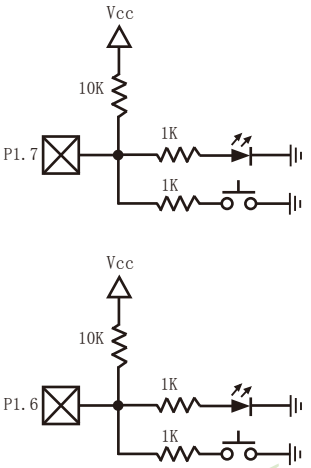
1、利用USB转串口连接电脑的RS485控制下载线路图(自动控制或I/O口控制)



2、利用RS232转串口连接电脑的RS485控制下载线路图(自动控制或I/O口控制)



附录F：一个I/O口驱动发光二极管并扫描按键



利用STC15系列单片机的I/O口可被设置成弱上拉,强上拉(推挽)输出,仅为输入(高阻),开漏等四种工作模式的特性,可以将STC15系列单片机的I/O口同时作为发光二极管驱动及按键检测用，这样可以大幅节省I/O口。

当驱动发光二极管时，将该I/O口设置成强推挽输出，输出高即可点亮发光二极管。

当检测按键时，将该I/O口设置成弱上拉输入，再读外部口的状态，即可检测按键。

STC MCU Limited

附录G：STC15系列单片机取代传统8051注意事项

STC15系列单片机的定时器0/定时器1与传统8051完全兼容，上电复位后，定时器部分缺省还是除12再计数的,所以定时器完全兼容。

STC15系列单片机对传统8051的111条指令执行速度全面提速,最快的指令快24倍,最慢的指令快3倍.靠软件延时实现精确延时的程序需要调整。

其它需注意的细节：

普通I/O口既作为输入又作为输出:

传统8051单片机执行I/O口操作,由高变低或由低变高,以及读外部状态都是12个时钟,而现在STC15系列单片机执行相应的操作是4个时钟.传统8051单片机如果对外输出为低,直接读外部状态是读不对的.必须先将I/O口置高才能够读对,而传统8051单片机由低变高的指令是12个时钟,该指令执行完成后,该I/O口也确实已变高.故可以紧跟着由低变高的指令后面,直接执行读该I/O口状态指令.而STC15系列单片机由于执行由低变高的指令是4个时钟,太快了,相应的指令执行完以后,I/O口还没有变高,要再过一个时钟之后,该I/O口才可以变高.故建议此状况下增加2个空操作延时指令再读外部口的状态。

I/O口驱动能力:

最新STC15系列单片机I/O口的灌电流是20mA,驱动能力超强,驱动大电流时,不容易烧坏.

传统STC89Cxx系列单片机I/O口的灌电流是6mA,驱动能力不够强,不能驱动大电流,建议使用STC15系列

看门狗:

最新STC15系列单片机的看门狗寄存器WDT_CONTR的地址在C1H,增加了看门狗复位标志位

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset value
WDT_CONTR	C1h	Watch-Dog-Timer Control register	WDT_FLAG	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0	xx00,0000

传统STC89系列增强型单片机看门狗寄存器WDT_CONTR的地址在E1H,没有看门狗复位标志位

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset value
WDT_CONTR	E1h	Watch-Dog-Timer Control register	-	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0	xx00,0000

最新STC15系列单片机的看门狗在ISP烧录程序可设置上电复位后直接启动看门狗,而传统STC89系列单片机无此功能.故最新STC15系列单片机看门狗更可靠.

STC MCU Limited

与EEPROM操作相关的寄存器

STC15Fxx单片机ISP/IAP控制寄存器地址和STC89xx系列单片机ISP/IAP控制寄存器地址不同如下：

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
STC15Fxx 系列 IAP_DATA STC89xx 系列 ISP_DATA	C2h E2h	ISP/IAP Flash Data Register									1111,1111
STC15Fxx 系列 IAP_ADDRH STC89xx 系列 ISP_ADDRH	C3h E3h	ISP/IAP Flash Address High									0000,0000
STC15Fxx 系列 IAP_ADDRL STC89xx 系列 ISP_ADDRL	C4h E4h	ISP/IAP Flash Address Low									0000,0000
STC15Fxx 系列 IAP_CMD STC89xx 系列 ISP_CMD	C5h E5h	ISP/IAP Flash Command Register	-	-	-	-	-	-	MS1	MS0	xxxx,xx00
STC15Fxx 系列 IAP_TRIG STC89xx 系列 ISP_TRIG	C6h E6h	ISP/IAP Flash Command Trigger									xxxx,xxxx
STC15Fxx系列 IAP_CONTR STC89xx 系列 ISP_CONTR	C7h E7h	ISP/IAP Control Register	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0	0000,x000

ISP/IAP_TRIG寄存器有效启动IAP操作,需顺序送入的数据不一样:

STC15系列单片机的ISP/IAP命令要生效,要对IAP_TRIG寄存器按顺序先送5Ah,再送A5h方可

STC89xx 系列单片机的ISP/IAP命令要生效,要对IAP_TRIG寄存器按顺序先送46h,再送B9h方可

EEPROM起始地址不一样:

STC15系列单片机的EEPROM起始地址全部从0000h开始,每个扇区512字节

STC89xx系列单片机的EEPROM起始地址分别有从1000h/2000h/4000h/8000h开始的,程序兼容性不够好.

STC MCU Limited

外部中断:

最新STC15系列单片机有5个外部中断。其中外部中断0(INT0)和外部中断1(INT1)可配置为2种中断触发方式:

第一种方式，仅下降沿触发中断，与传统8051的外部中断0和1的下降沿中断兼容。

第二种方式，上升沿中断和下降沿中断同时支持。

另外相对传统STC89系列单片机,最新的STC15系列单片机还增加了外部中断2、外部中断3和外部中断4，这三个新增的外部中断都只能下降沿触发中断。

而传统STC89系列单片机的外部中断0和外部中断1只可以配置为下降沿中断或低电平中断。

定时器:

最新STC15系列单片机的定时器/计数器0和定时器/计数器1与传统STC89系列单片机的定时器/计数器0和定时器/计数器1的最大不同在于定时器的工作模式0。最新STC15系列单片机的定时器/计数器0和定时器/计数器1的工作模式0是16位自动重装载模式，而传统STC89系列单片机的定时器/计数器0和定时器/计数器1的模式0是13位定时/计数器模式。最新STC15系列单片机的定时器/计数器0和定时器/计数器1仍保留着其他3种工作模式，这3中工作模式与传统的STC89系列单片机的定时器/计数器0和定时器/计数器1的工作模式兼容。另外传统的STC89系列单片机还设有定时器2，而最新STC15系列单片机只有定时器0和1。

外部时钟和内部时钟:

最新STC15系列单片机内部集成了高精度R/C振荡器作为系统时钟，省掉了昂贵的外部晶体振荡时钟。而传统STC89系列单片机只能使用外部晶体或时钟作为系统时钟.

功耗:

功耗由2部分组成,晶体振荡器放大电路的功耗和单片机的数字电路功耗组成,

晶体振荡器放大电路的功耗: 最新STC15系列单片机比STC89xx系列低.

单片机的数字电路功耗:时钟频率越高,功耗越大,最新STC15系列单片机在相同工作频率下,指令执行速度比传统STC89系列单片机快3-24倍,故可用较低的时钟频率工作,这样功耗更低.而且STC15系列单片机可以利用内部的时钟分频器对时钟进行分频,以较低的频率工作,使得单片机的功耗更低。

掉电唤醒:

最新STC15系列单片机支持外部中断上升沿或下降沿均可唤醒，也可仅下降沿唤醒。传统STC89系列单片机是只支持外部中断低电平唤醒。另外最新STC15系列单片机还内置了掉电唤醒专用定时器

附录H：STC15系列对指令系统的提升

----与普通8051指令代码完全兼容，但执行的时间效率大幅提升
----其中INC DPTR和MUL AB指令的执行速度大幅提升24倍
----共有12条指令，一个时钟就可以执行完成，平均速度快8~12倍

如果按功能分类，STC15系列单片机指令系统可分为：
1. 算术操作类指令；
2. 逻辑操作类指令；
3. 数据传送类指令；
4. 布尔变量操作类指令；
5. 控制转移类指令。

按功能分类的指令系统表如下表所示。

指令执行速度效率提升总结(新15系列)：

指令系统共包括111条指令，其中：

执行速度快24倍的	共2条
执行速度快12倍的	共28条
执行速度快8倍的	共19条
执行速度快6倍的	共40条
执行速度快4.8倍的	共8条
执行速度快4倍的	共14条

根据对指令的使用频率分析统计，STC15系列 1T 的8051单片机比普通的8051单片机在同样的工作频率下运行速度提升了8～12倍。

指令执行时钟数统计（供参考）(新15系列)：

指令系统共包括111条指令，其中：

1个时钟就可执行完成的指令	共22条
2个时钟就可执行完成的指令	共37条
3个时钟就可执行完成的指令	共31条
4个时钟就可执行完成的指令	共12条
5个时钟就可执行完成的指令	共8条
6个时钟就可执行完成的指令	共1条

111条指令全部执行完一遍所需的时钟为283个时钟。

STC MCU Limited

现STC15系列单片机采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度又比STC早期的1T系列单片机(如STC12系列/STC11系列/STC10系列)的速度快20%.

算术操作类指令

助记符		功能说明	字节数	传统8051单片机所需时钟	STC15系列单片机所需时钟（采用STC-Y5超高速 1T 8051 内核）	效率提升
ADD	A，Rn	寄存器内容加到累加器	1	12	1	12倍
ADD	A，direct	直接地址单元中的数据加到累加器	2	12	2	6倍
ADD	A，@Ri	间接RAM中的数据加到累加器	1	12	2	6倍
ADD	A，#data	立即数加到累加器	2	12	2	6倍
ADDC	A，Rn	寄存器带进位加到累加器	1	12	1	12倍
ADDC	A，direct	直接地址单元的内容带进位加到累加器	2	12	2	6倍
ADDC	A，@Ri	间接RAM内容带进位加到累加器	1	12	2	6倍
ADDC	A，#data	立即数带进位加到累加器	2	12	2	6倍
SUBB	A，Rn	累加器带借位减寄存器内容	1	12	1	6倍
SUBB	A，direct	累加器带借位减直接地址单元的内容	2	12	2	6倍
SUBB	A，@Ri	累加器带借位减间接RAM中的内容	1	12	2	6倍
SUBB	A，#data	累加器带借位减立即数	2	12	2	6倍
INC	A	累加器加1	1	12	1	12倍
INC	Rn	寄存器加1	1	12	2	6倍
INC	direct	直接地址单元加1	2	12	3	4倍
INC	@Ri	间接RAM单元加1	1	12	3	4倍
DEC	A	累加器减1	1	12	1	12倍
DEC	Rn	寄存器减1	1	12	2	6倍
DEC	direct	直接地址单元减1	2	12	3	4倍
DEC	@Ri	间接RAM单元减1	1	12	3	4倍
INC	DPTR	地址寄存器DPTR加1	1	24	1	24倍
MUL	AB	A乘以B	1	48	2	24倍
DIV	AB	A除以B	1	48	6	8倍
DA	A	累加器十进制调整	1	12	3	4倍

STC MCU Limited

逻辑操作类指令

助记符		功能说明	字节数	传统8051单片机所需时钟	STC15系列单片机所需时钟（采用STC-Y5超高速 1T 8051 内核）	效率提升
ANL	A，Rn	累加器与寄存器相“与”	1	12	1	12倍
ANL	A，direct	累加器与直接地址单元相“与”	2	12	2	6倍
ANL	A，@Ri	累加器与间接RAM单元相“与”	1	12	2	6倍
ANL	A，#data	累加器与立即数相“与”	2	12	2	6倍
ANL	direct， A	直接地址单元与累加器相“与”	2	12	3	4倍
ANL	direct，#data	直接地址单元与立即数相“与”	3	24	3	8倍
ORL	A， Rn	累加器与寄存器相“或”	1	12	1	12倍
ORL	A，direct	累加器与直接地址单元相“或”	2	12	2	6倍
ORL	A, @Ri	累加器与间接RAM单元相“或”	1	12	2	6倍
ORL	A，# data	累加器与立即数相“或”	2	12	2	6倍
ORL	direct， A	直接地址单元与累加器相“或”	2	12	3	4倍
ORL	direct， #data	直接地址单元与立即数相“或”	3	24	3	8倍
XRL	A， Rn	累加器与寄存器相“异或”	1	12	1	12倍
XRL	A， direct	累加器与直接地址单元相“异或”	2	12	2	6倍
XRL	A， @Ri	累加器与间接RAM单元相“异或”	1	12	2	6倍
XRL	A， # data	累加器与立即数相“异或”	2	12	2	6倍
XRL	direct， A	直接地址单元与累加器相“异或”	2	12	3	4倍
XRL	direct，#data	直接地址单元与立即数相“异或”	3	24	3	8倍
CLR	A	累加器清“0”	1	12	1	12倍
CPL	A	累加器求反	1	12	1	12倍
RL	A	累加器循环左移	1	12	1	12倍
RLC	A	累加器带进位位循环左移	1	12	1	12倍
RR	A	累加器循环右移	1	12	1	12倍
RRC	A	累加器带进位位循环右移	1	12	1	12倍
SWAP	A	累加器内高低半字节交换	1	12	1	12倍

STC MCU Limited

数据传送类指令

助记符		功能说明	字节数	传统8051单片机所需时钟	STC15系列单片机所需时钟（采用STC-Y5超高速 1T 8051 内核）	效率提升
MOV	A, Rn	寄存器内容送入累加器	1	12	1	12倍
MOV	A, direct	直接地址单元中的数据送入累加器	2	12	2	6倍
MOV	A, @Ri	间接RAM中的数据送入累加器	1	12	2	6倍
MOV	A, #data	立即数送入累加器	2	12	2	6倍
MOV	Rn, A	累加器内容送入寄存器	1	12	1	12倍
MOV	Rn, direct	直接地址单元中的数据送入寄存器	2	24	3	8倍
MOV	Rn, #data	立即数送入寄存器	2	12	2	6倍
MOV	direct, A	累加器内容送入直接地址单元	2	12	2	6倍
MOV	direct, Rn	寄存器内容送入直接地址单元	2	24	2	12倍
MOV	direct, direct	直接地址单元中的数据送入另一个直接地址单元	3	24	3	8倍
MOV	direct, @Ri	间接RAM中的数据送入直接地址单元	2	24	3	8倍
MOV	direct, #data	立即数送入直接地址单元	3	24	3	8倍
MOV	@Ri, A	累加器内容送间接RAM单元	1	12	2	6倍
MOV	@Ri, direct	直接地址单元数据送入间接RAM单元	2	24	3	8倍
MOV	@Ri, #data	立即数送入间接RAM单元	2	12	2	6倍
MOV	DPTR,#data16	16位立即数送入数据指针	3	24	3	8倍
MOVC	A, @A+DPTR	以DPTR为基地址变址寻址单元中的数据送入累加器	1	24	5	4.8倍
MOVC	A, @A+PC	以PC为基地址变址寻址单元中的数据送入累加器	1	24	4	6倍
MOVX	A, @Ri	将逻辑上在片外、物理上在片内的扩展RAM(8位地址)的内容送入累加器A中，读操作	1	24	3	8倍
MOVX	@Ri, A	将累加器A的内容送入逻辑上在片外、物理上在片内的扩展RAM(8位地址)中，写操作	1	24	4	8倍
MOVX	A, @DPTR	将逻辑上在片外、物理上在片内的扩展RAM(16位地址)的内容送入累加器A中，读操作	1	24	2	12倍
MOVX	@DPTR, A	将累加器A的内容送入逻辑上在片外、物理上在片内的扩展RAM(16位地址)中，写操作	1	24	3	8倍
MOVX	A, @Ri	将逻辑上在片外、物理上也在片外的扩展RAM(8位地址)的内容送入累加器A中，读操作	1	24	5xN+2 N的取值见下列说明	*Note1
MOVX	@Ri, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(8位地址)中，写操作	1	24	5×N+3 N的取值见下列说明	*Note1
MOVX	A, @DPTR	将逻辑上在片外、物理上也在片外的扩展RAM(16位地址)的内容送入累加器A中，读操作	1	24	5×N+1 N的取值见下列说明	*Note1
MOVX	@DPTR, A	将累加器A的内容送入逻辑上在片外、物理上也在片外的扩展RAM(16位地址)中，写操作	1	24	5×N+2 N的取值见下列说明	*Note1
PUSH	direct	直接地址单元中的数据压入堆栈	2	24	3	8倍
POP	direct	栈底数据弹出送入直接地址单元	2	24	2	12倍
XCH	A, Rn	寄存器与累加器交换	1	12	2	6倍
XCH	A,direct	直接地址单元与累加器交换	2	12	3	4倍
XCH	A, @Ri	间接RAM与累加器交换	1	12	3	4倍
XCHD	A, @Ri	间接RAM的低半字节与累加器交换	1	12	3	4倍

当EXRTS[1:0] = [0,0]时， 表中N=1；
当EXRTS[1:0] = [0,1]时， 表中N=2；
当EXRTS[1:0] = [1,0]时， 表中N=4；
当EXRTS[1:0] = [1,1]时， 表中N=8.

EXRTS[1：0]为寄存器BUS_SPEED中的B1,B0位

布尔变量操作类指令

助记符		功能说明	字节数	传统8051单片机所需时钟	STC15系列单片机所需时钟(采用STC-Y5超高速 1T 8051 内核)	效率提升
CLR	C	清零进位位	1	12	1	12倍
CLR	bit	清0直接地址位	2	12	3	4倍
SETB	C	置1进位位	1	12	1	12倍
SETB	bit	置1直接地址位	2	12	3	4倍
CPL	C	进位位求反	1	12	1	12倍
CPL	bit	直接地址位求反	2	12	3	4倍
ANL	C, bit	进位位和直接地址位相“与”	2	24	2	12倍
ANL	C, /bit	进位位和直接地址位的反码相“与”	2	24	2	12倍
ORL	C, bit	进位位和直接地址位相“或”	2	24	2	12倍
ORL	C, /bit	进位位和直接地址位的反码相“或”	2	24	2	12倍
MOV	C, bit	直接地址位送入进位位	2	12	2	12倍
MOV	bit, C	进位位送入直接地址位	2	24	3	8倍
JC	rel	进位位为1则转移	2	24	3	8倍
JNC	rel	进位位为0则转移	2	24	3	8倍
JB	bit, rel	直接地址位为1则转移	3	24	5	4.8倍
JNB	bit, rel	直接地址位为0则转移	3	24	5	4.8倍
JBC	bit, rel	直接地址位为1则转移，该位清0	3	24	5	4.8倍

STC MCU Limited

控制转移类指令

助记符	功能说明	字节数	传统8051单片机所需时钟	STC15系列单片机所需时钟(采用STC-Y5超高速 1T 8051 内核)	效率提升
ACALL addr11	绝对（短）调用子程序	2	24	4	6倍
LCALL addr16	长调用子程序	3	24	4	6倍
RET	子程序返回	1	24	4	6倍
RETI	中断返回	1	24	4	6倍
AJMP addr11	绝对（短）转移	2	24	3	8倍
LJMP addr16	长转移	3	24	4	6倍
SJMP re1	相对转移	2	24	3	8倍
JMP @A+DPTR	相对于DPTR的间接转移	1	24	5	4.8倍
JZ re1	累加器为零转移	2	24	4	6倍
JNZ re1	累加器非零转移	2	24	4	6倍
CJNE A，direct，re1	累加器与直接地址单元比较，不相等则转移	3	24	5	4.8倍
CJNE A，#data，re1	累加器与立即数比较，不相等则转移	3	24	4	6倍
CJNE Rn，#data，re1	寄存器与立即数比较，不相等则转移	3	24	4	6倍
CJNE @Ri，#data，re1	间接RAM单元与立即数比较，不相等则转移	3	24	5	4.8倍
DJNZ Rn，re1	寄存器减1，非零转移	2	24	4	6倍
DJNZ direct，re1	直接地址单元减1，非零转移	3	24	5	4.8倍
NOP	空操作	1	12	1	12倍

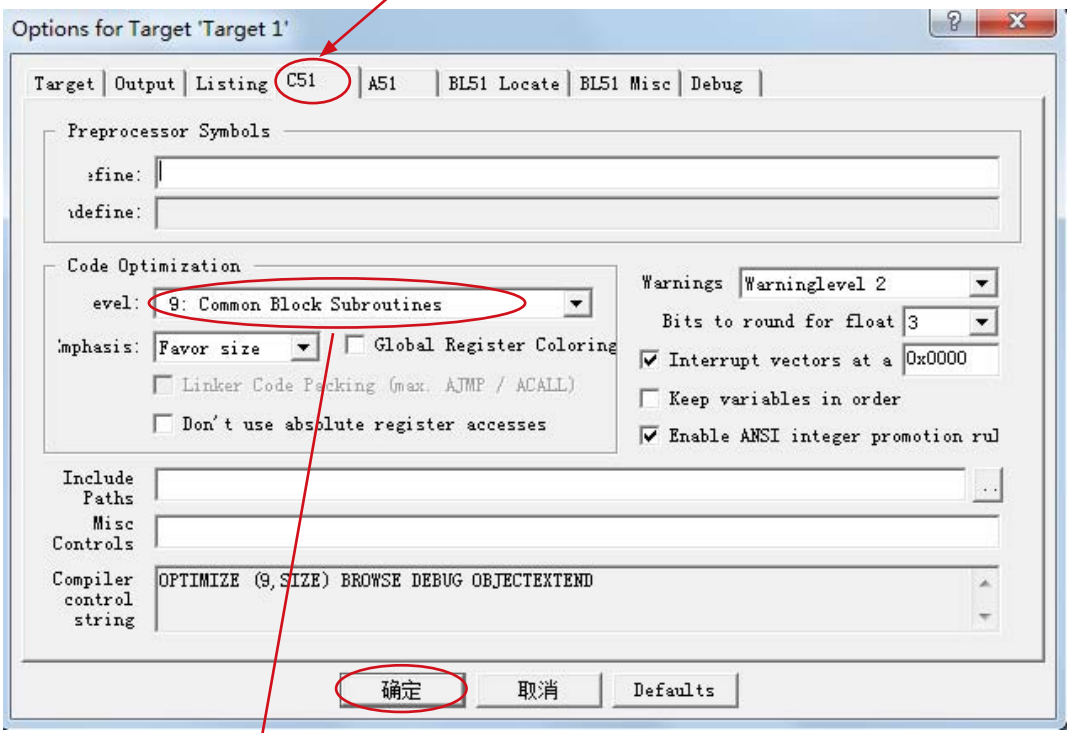
本次指令系统总结更新于2011-10-17日止

STC MCU Limited

附录I：如何利用Keil C软件减少代码长度

在Keil C软件中选择作如下设置，能将原代码长度最大减少10K。

1. 在“Project”菜单中选择“Options for Target”
2. 在“Options for Target”中选择“C51”



3. 选择按空间大小，9级优化程序
4. 点击“确定”后，重新编译程序即可。

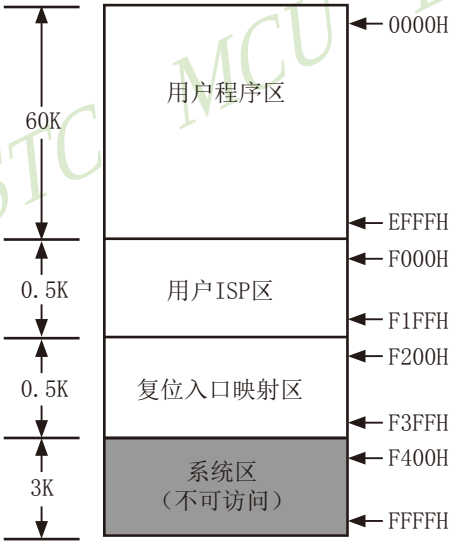
附录J：使用STC的IAP系列单片机开发自己的ISP程序

——基于IAP15F2K61S2单片机

随着IAP（In-Application-Programming）技术在单片机领域的不断发展，给应用系统程序代码升级带来了极大的方便。STC的串口ISP（In-System-Programming）程序就是使用IAP功能来对用户的程序进行在线升级的，但是出于对用户代码的安全着想，底层代码和上层应用程序都没有开源，为此STC推出了IAP系列单片机，即整颗MCU的Flash空间，用户均可在自己的程序中进行改写，从而使得有用户需要开发字节的ISP程序的想法得以实现。本文以STC的IAP15F2K61S2为例，详细说明了使用STC的IAP单片机开发用户自己的ISP程序的方法，并给出了基于Keil环境的汇编和C源码。

一．内部FLASH规划

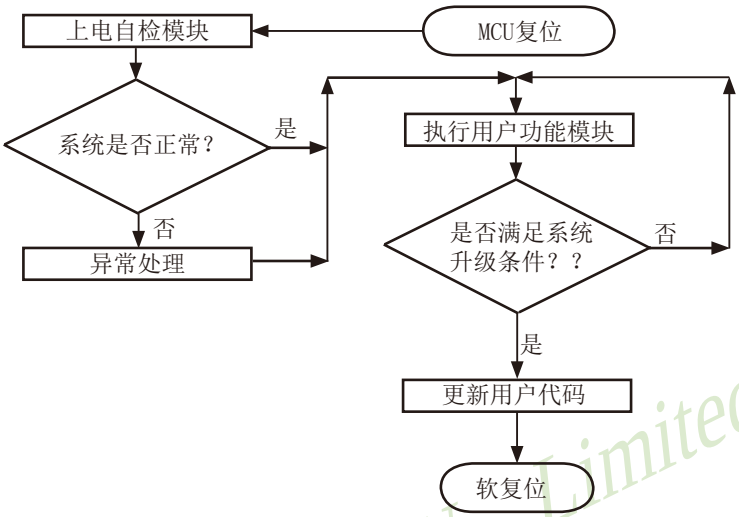
示例单片机使用IAP15F2K61S2，用户可以使用的最大程序空间为60K字节，整个Flash空间划分如下：



FLASH空间中，从地址0000H开始的连续60K字节的空间为用户程序区。当满足特定的下载条件时，用户需要自行将程序跳转到用户ISP程序区，此时可对用户程序区进行擦除和改写，以达到更新用户程序的目的。

STC MCU Limited

二．程序的基本框架



三．下位机固件程序说明

下位机固件程序包括两部分：ISP（ISP代码）和AP（用户代码）

ISP代码（汇编语言）如下：

```
;-------------------------------------------------------------------------------------
; --- STC MCU International Limited ----------------------------------------------
; --- STC IAP 系列单片机实现用户ISP 演示程序 -------------------------------
; --- Mobile: (86)13922805190 -----------------------------------------------------
; --- Fax: 86-755-82944243 -------------------------------------------------------
; --- Tel: 86-755-82948412 -------------------------------------------------------
; --- Web: www.STCMCU.com --------------------------------------------------------
; 如果要在程序中使用或者在文章中引用该程序,请在程序中或文章中注明
; 使用了宏晶科技的资料或程序
;-------------------------------------------------------------------------------------

;-------------------------------------------------------------------------------------
;定义常数

UARTBAUD        EQU     0FFE8H          ;定义串口波特率发生器的重载值
                                        ;(65536-11059200/4/115200)

;ENABLE_IAP     EQU     80H             ;系统工作频率<30MHz
;ENABLE_IAP     EQU     81H             ;系统工作频率<24MHz
```

STC MCU Limited

```
;ENABLE_IAP    EQU    82H          ;系统工作频率<20MHz
ENABLE_IAP     EQU    83H          ;系统工作频率<12MHz
;ENABLE_IAP    EQU    84H          ;系统工作频率<6MHz
;ENABLE_IAP    EQU    85H          ;系统工作频率<3MHz
;ENABLE_IAP    EQU    86H          ;系统工作频率<2MHz
;ENABLE_IAP    EQU    87H          ;系统工作频率<1MHz

;-----------------------------------------------
;定义特殊功能寄存器

AUXR           DATA   08EH         ;附件功能控制寄存器
WDT_CONTR      DATA   0C1H         ;看门狗控制寄存器
IAP_DATA       DATA   0C2H         ;IAP数据寄存器
IAP_ADDRH      DATA   0C3H         ;IAP高地址寄存器
IAP_ADDRL      DATA   0C4H         ;IAP低地址寄存器
IAP_CMD        DATA   0C5H         ;IAP命令寄存器
IAP_TRIG       DATA   0C6H         ;IAP命令触发寄存器
IAP_CONTR      DATA   0C7H         ;IAP控制寄存器

;-----------------------------------------------
;定义ISP模块使用的变量

ISPCODE        EQU    0F000H       ;ISP模块入口地址(1页),同时也是下载接口地址
APENTRY        EQU    0F200H       ;应用程序入口地址数据(1页)

;-----------------------------------------------

        ORG    0000H

        LJMP   ISP_ENTRY           ;系统复位入口

RESET:
        MOV    SCON,   #50H                ;设置串口模式(8位数据位,波特率可变,无校验位)
        MOV    AUXR,   #40H                ;T1工作于1T模式
        MOV    TH1,    #HIGH UARTBAUD      ;波特率设置
        MOV    TL1,    #LOW UARTBAUD       ;
        SETB   TR1                         ;启动定时器1
NEXT1:
        MOV    R0,     #16
NEXT2:
        JNB    RI,     $                   ;等待串口数据
        CLR    RI
        MOV    A,      SBUF
        CJNE   A,      #7FH,NEXT1          ;判断是否为7F
```

STC MCU Limited

```
        DJNZ    R0,     NEXT2
        LJMP    ISPPROGRAM              ;跳转到下载界面

;-------------------------------------------------
;ISP功能模块
;包括上电自检模块和代码更新模块
;-------------------------------------------------

        ORG     ISPCODE

ISPPROGRAM:
        CLR     A
        MOV     PSW,    A               ;ISP模块使用第0组寄存器
        MOV     IE,     A               ;关闭所有中断
        CLR     RI                      ;清除串口接收标志
        SETB    TI                      ;置串口发送标志
        CLR     TR0
        MOV     SP,     #5FH            ;设置堆栈指针

;-------------------------------------------------

        MOV     A,      #5AH            ;返回5A 55到PC,表示ISP擦除模块已准备就绪
        CALL    ISP_SENDUART
        MOV     A,      #055H
        CALL    ISP_SENDUART
        CALL    ISP_RECVACK             ;接收应答数据

        MOV     IAP_ADDRL,      #0      ;首先在第2页起始地址写 "LJMP ISP_ENTRY"指令
        MOV     IAP_ADDRH,      #02H
        CALL    ISP_ERASEIAP
        MOV     A,      #02H
        CALL    ISP_PROGRAMIAP                  ;编程用户代码复位向量代码
        MOV     A,      #HIGH ISP_ENTRY
        CALL    ISP_PROGRAMIAP                  ;编程用户代码复位向量代码
        MOV     A,      #LOW ISP_ENTRY
        CALL    ISP_PROGRAMIAP                  ;编程用户代码复位向量代码

        MOV     IAP_ADDRL,      #0              ;用户代码地址从0开始
        MOV     IAP_ADDRH,      #0
        CALL    ISP_ERASEIAP
        MOV     A,      #02H
        CALL    ISP_PROGRAMIAP                  ;编程用户代码复位向量代码
        MOV     A,      #HIGH ISP_ENTRY
        CALL    ISP_PROGRAMIAP                  ;编程用户代码复位向量代码
```

STC MCU Limited

```
        MOV     A,      #LOW ISP_ENTRY
        CALL    ISP_PROGRAMIAP                  ;编程用户代码复位向量代码

        MOV     IAP_ADDRL,      #0              ;新代码缓冲区地址
        MOV     IAP_ADDRH,      #02H
        MOV     R7,     #119                    ;擦除60K-512字节
ISP_ERASEAP:
        CALL    ISP_ERASEIAP
        INC     IAP_ADDRH                       ;目标地址+512
        INC     IAP_ADDRH
        DJNZ    R7,     ISP_ERASEAP             ;判断是否擦除完成

        MOV     IAP_ADDRL,      #LOW APENTRY    ;用户代码复位入口页
        MOV     IAP_ADDRH,      #HIGH APENTRY
        CALL    ISP_ERASEIAP

        MOV     A,      #5AH                    ;返回5A A5到PC,表示ISP编程模块已准备就绪
        CALL    ISP_SENDUART
        MOV     A,      #0A5H
        CALL    ISP_SENDUART
        CALL    ISP_RECVACK                     ;接收应答数据

        CALL    ISP_RECVUART                    ;接收长度高字节
        MOV     R0,     A
        CALL    ISP_RECVUART                    ;接收长度低字节
        MOV     R1,     A
        CLR     C                               ;将(总长度-3)的补码存入DPTR
        MOV     A,      #03H
        SUBB    A,      R1
        MOV     DPL,    A
        CLR     A
        SUBB    A,      R0
        MOV     DPH,    A

        CALL    ISP_RECVUART                    ;映射用户代码复位入口代码到映射区
        CALL    ISP_PROGRAMIAP                  ;0000
        CALL    ISP_RECVUART
        CALL    ISP_PROGRAMIAP                  ;0001
        CALL    ISP_RECVUART
        CALL    ISP_PROGRAMIAP                  ;0002

        MOV     IAP_ADDRL,      #03H            ;用户代码起始地址
        MOV     IAP_ADDRH,      #00H
ISP_PROGRAMNEXT:
```

STC MCU Limited

```
        CALL    ISP_RECVUART                ;接收代码数据
        CALL    ISP_PROGRAMIAP              ;编程到用户代码区
        INC     DPTR
        MOV     A,      DPL
        ORL     A,      DPH
        JNZ     ISP_PROGRAMNEXT             ;长度检测

ISP_SOFTRESET:
        MOV     IAP_CONTR,      #20H        ;软件复位系统
        SJMP    $

;-------------------------------------------------
;用户ISP主程序的入口地址
;-------------------------------------------------
ISP_ENTRY:
        MOV     IAP_ADDRL,      #00H        ;电压测试模块
        MOV     IAP_ADDRH,      #0F0H       ;测试地址F000H
        MOV     IAP_DATA,       #53H        ;测试数据1
        CALL    ISP_READIAP
        XRL     A,              #53H        ;测试是否可以读出数据
        JZ      ISP_ENTRY                   ;若读不出数据,则需等待电压稳定
        INC     IAP_ADDRL                   ;测试地址F001H
        MOV     IAP_DATA,       #45H        ;测试数据2
        CALL    ISP_READIAP
        XRL     A,              #45H        ;测试是否可以读出数据
        JZ      ISP_ENTRY                   ;若读不出数据,则需等待电压稳定

        JMP     GOTOAP                      ;电压稳定后开始运行用户代码
        MOV     TMOD,           #00H
        MOV     SCON,           #5AH        ;设置串口模式(8位数据位,波特率可变,无校验位)
        MOV     AUXR,           #40H                ;T1工作于1T模式
        MOV     TH1,            #HIGH UARTBAUD      ;波特率设置
        MOV     TL1,            #LOW UARTBAUD
        SETB    TR1                                 ;启动定时器1

        MOV     R0,             #16
ISP_CHECKNEXT:
        CALL    ISP_RECVUART                        ;接收同步数据
        JC      GOTOAP
        CJNE    A,              #7FH,GOTOAP         ;判断是否为7F
        DJNZ    R0,             ISP_CHECKNEXT
        MOV     A,              #5AH                ;返回5A 69到PC,表示ISP模块已准备就绪
        CALL    ISP_SENDUART
        MOV     A,              #69H
```

STC MCU Limited

```
        CALL    ISP_SENDUART
        CALL    ISP_RECVACK                 ;接收应答数据
        LJMP    ISPPROGRAM                  ;跳转到下载界面

;-----------------------------------------------
;跳转到用户程序区,开始正常运行用户程序
;-----------------------------------------------
GOTOAP:
        CLR     A                           ;将SFR恢复为复位值
        MOV     TCON,   A
        MOV     TMOD,   A
        MOV     TL0,    A
        MOV     TH0,    A
        MOV     TL1,    A
        MOV     TH1,    A
        MOV     SCON,   A
        MOV     AUXR,   A
        LJMP    APENTRY                     ;正常运行用户程序

;-----------------------------------------------
;接收来自于上位机的串口应答数据
;-----------------------------------------------
ISP_RECVACK:
        CALL    ISP_RECVUART
        JC      GOTOAP
        XRL     A,      #7FH
        JZ      ISP_RECVACK                 ;跳过同步数据 (7FH)
        CJNE    A,      #25H,   GOTOAP      ;应答数据1检测(5AH)
        CALL    ISP_RECVUART
        JC      GOTOAP
        XRL     A,      #69H                ;应答数据2检测(69H)
        JNZ     GOTOAP
        RET

;-----------------------------------------------
;接收1字节串口数据
;出口参数: ACC (接收到的数据)
;出口参数: C (1:超时)
;-----------------------------------------------
ISP_RECVUART:
        CLR     TR0
        CLR     A
        MOV     TL0,    A                   ;初始化超时定时器
        MOV     TH0,    A
```

STC MCU Limited

```
        CLR     TF0
        SETB    TR0
        MOV     WDT_CONTR,    #17H                ;清看门狗
ISP_RECVWAIT:
        JBC     TF0,    ISP_RECVTIMEOUT           ;超时检测
        JNB     RI,     ISP_RECVWAIT              ;等待接收完成
        MOV     A,      SBUF                      ;读取串口数据
        CLR     RI                                ;清除标志
        CLR     C                                 ;正确接收串口数据
        RET
ISP_RECVTIMEOUT:
        SETB    C                                 ;超时退出
        RET

;-------------------------------------------
;发送1字节串口数据
;入口参数: ACC (待发送的数据)
;-------------------------------------------
ISP_SENDUART:
        MOV     WDT_CONTR,    #17H                ;清看门狗
        JNB     TI,     ISP_SENDUART              ;等待前一个数据发送完成
        CLR     TI                                ;清除标志
        MOV     SBUF,   A                         ;发送当前数据
        RET

;-------------------------------------------
;擦除IAP扇区
;-------------------------------------------
ISP_ERASEIAP:
        MOV     WDT_CONTR,    #17H                ;清看门狗
        MOV     IAP_CONTR,    #ENABLE_IAP         ;使能IAP功能
        MOV     IAP_CMD,      #3                  ;擦除命令
        MOV     IAP_TRIG,     #5AH                ;触发ISP命令
        MOV     IAP_TRIG,     #0A5H
        RET

;-------------------------------------------
;编程IAP字节
;入口参数: ACC (待编程的数据)
;-------------------------------------------
ISP_PROGRAMIAP:
        MOV     WDT_CONTR,    #17H                ;清看门狗
        MOV     IAP_CONTR,    #ENABLE_IAP         ;使能IAP功能
        MOV     IAP_CMD,      #2                  ;编程命令
```

STC MCU Limited

```
        MOV     IAP_DATA,       A                   ;将当前数据送IAP数据寄存器
        MOV     IAP_TRIG,       #5AH                ;触发ISP命令
        MOV     IAP_TRIG,       #0A5H
        MOV     A,              IAP_ADDRL           ;IAP地址+1
        ADD     A,              #01H
        MOV     IAP_ADDRL,      A
        MOV     A,              IAP_ADDRH
        ADDC    A,              #00H
        MOV     IAP_ADDRH,      A
        RET

;-------------------------------------------------
;读取IAP字节
;出口参数: ACC (读出的数据)
;-------------------------------------------------
ISP_READIAP:
        MOV     IAP_CONTR,      #ENABLE_IAP         ;使能IAP功能
        MOV     IAP_CMD,        #1                  ;读命令
        MOV     IAP_TRIG,       #5AH                ;触发ISP命令
        MOV     IAP_TRIG,       #0A5H
        MOV     A,          IAP_DATA
        RET

;-------------------------------------------------

        ORG     APENTRY
        LJMP    RESET

;-------------------------------------------------

        END
```

ISP代码包括如下外部接口模块：

ISPPROGRAM：程序下载入口地址，绝对地址F000H

ISP_ENTRY：上电系统自检程序（系统自动调用）

对于用户程序而言，用户只需要在满足下载条件时，将PC值跳转到ISPPROGRAM （即F000H的绝对地址），即可实现代码更新。

STC MCU Limited

用户代码（C语言及汇编语言）示例如下：

1、C语言用户代码

```
/*---------------------------------------------------------------------------------------*/
/* --- STC MCU International Limited ---------------------------------------------------*/
/* --- STC IAP 系列单片机实现用户ISP 演示程序 ------------------------------------------*/
/* --- Mobile: (86)13922805190 ---------------------------------------------------------*/
/* --- Fax: 86-755-82944243 ------------------------------------------------------------*/
/* --- Tel: 86-755-82948412 ------------------------------------------------------------*/
/* --- Web: www.STCMCU.com -------------------------------------------------------------*/
/* 如果要在程序中使用或者在文章中引用该程序,请在程序中或文章中注明  */
/* 使用了宏晶科技的资料或程序  ---------------------------------------------------- */
/*---------------------------------------------------------------------------------------*/

#include "reg51.h"

/* 定义常数 */
#define   FOSC              11059200L                   //系统时钟频率
#define   BAUD              115200                      //定义串口波特率
#define   RELOAD            (65536 - FOSC/4/BAUD)       //定时器重载值

#define   ISPPROGRAM()  ((void (code *)())0xF000)()     //ISP下载程序入口地址

/* 定义相关SFR */
sfr       AUXR    =   0x8E;                           //辅助寄存器

void uart() interrupt 4 using 1                 //串口中断服务程序
{
        static char cnt7f = 0;                  //7f的计数器

        if (TI) TI = 0;

        if (RI)
        {
                if (SBUF == 0x7f)
                {
                        if (++cnt7f >= 16)
                        {
                                ISPPROGRAM();       //调用下载模块(****重要语句****)
                        }
                }
                else
                {
                        cnt7f = 0;
```

STC MCU Limited

```
                }
                RI = 0;
        }
}

void main()
{
        SCON    =       0x50;                   //设置串口模式(8位数据位,波特率可变,无校验位)
        AUXR    =       0x40;                   //T1工作于1T模式
        TH1     =       RELOAD >> 8;            //波特率设置
        TL1     =       RELOAD;
        TR1     =       1;                      //启动定时器1
        ES      =       1;                      //使能串口中断
        EA      =       1;                      //打开全局中断开关

        while (1)
        {
                P1++;                           //用户示例代码
        }
}
```

2、汇编语言用户代码

```
;/*------------------------------------------------------------------------------------------------*/
;/* --- STC MCU International Limited ----------------------------------------------------------*/
;/* --- STC IAP 系列单片机实现用户ISP 演示程序 -------------------------------------------------*/
;/* --- Mobile: (86)13922805190 ----------------------------------------------------------------*/
;/* --- Fax: 86-755-82944243 -------------------------------------------------------------------*/
;/* --- Tel: 86-755-82948412 -------------------------------------------------------------------*/
;/* --- Web: www.STCMCU.com --------------------------------------------------------------------*/
;/* 如果要在程序中使用或者在文章中引用该程序,请在程序中或文章中注明  */
;/* 使用了宏晶科技的资料或程序 -------------------------------------------------------------*/
;/*------------------------------------------------------------------------------------------------*/

;-----------------------------------------------
;/*定义常数*/

UARTBAUD        EQU     0FFE8H          ;定义串口波特率 (65536-11059200/4/115200)
ISPPROGRAM      EQU     0F000H          ;ISP下载程序入口地址

;-----------------------------------------------
;/*定义特殊功能寄存器*/
```

STC MCU Limited

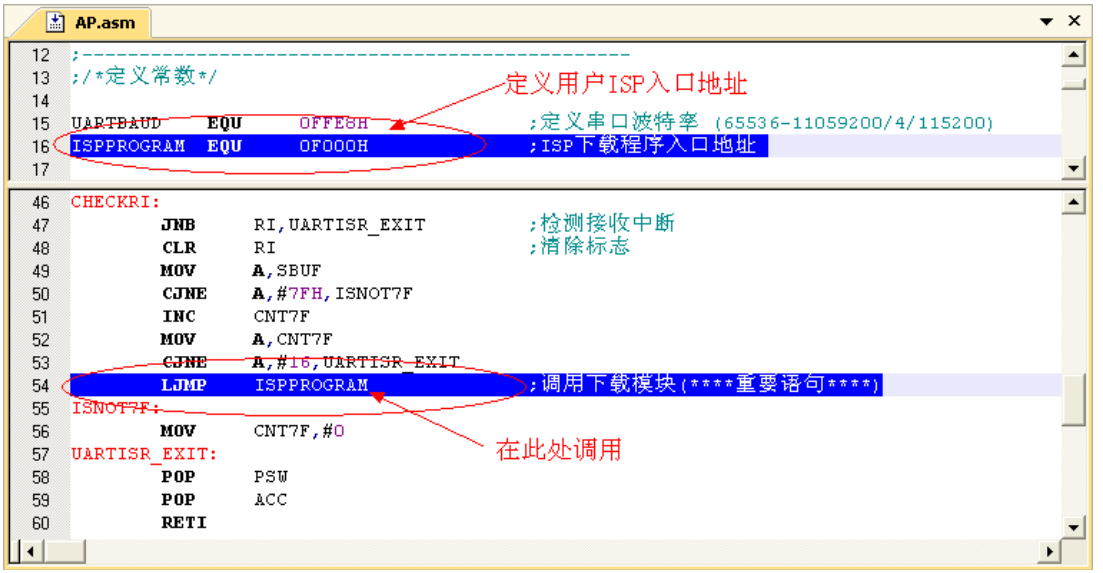
```
AUXR            DATA    08EH            ;附件功能控制寄存器

;-------------------------------------------------
;/*定义用户变量*/

CNT7F           DATA    60H             ;接收7F的计数器,当连续接收到16次7F后进入ISP下载模式

;-------------------------------------------------
;/*中断向量表*/

        ORG     0000H
        LJMP    START                   ;系统复位入口

        ORG     0023H
        LJMP    UART_ISR                ;串口中断入口

;-------------------------------------------------
;/*串口中断服务程序*/

UART_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     PSW,    #08H
        JNB     TI,     CHECKRI         ;检测发送中断
        CLR     TI                              ;清除标志
CHECKRI:
        JNB     RI,     UARTISR_EXIT            ;检测接收中断
        CLR     RI                              ;清除标志
        MOV     A,      SBUF
        CJNE    A,      #7FH,   ISNOT7F
        INC     CNT7F
        MOV     A,      CNT7F
        CJNE    A,      #16,    UARTISR_EXIT
        LJMP    ISPPROGRAM                      ;调用下载模块(****重要语句****)
ISNOT7F:
        MOV     CNT7F,  #0
UARTISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

;-------------------------------------------------
;/*主程序入口*/
```

STC MCU Limited

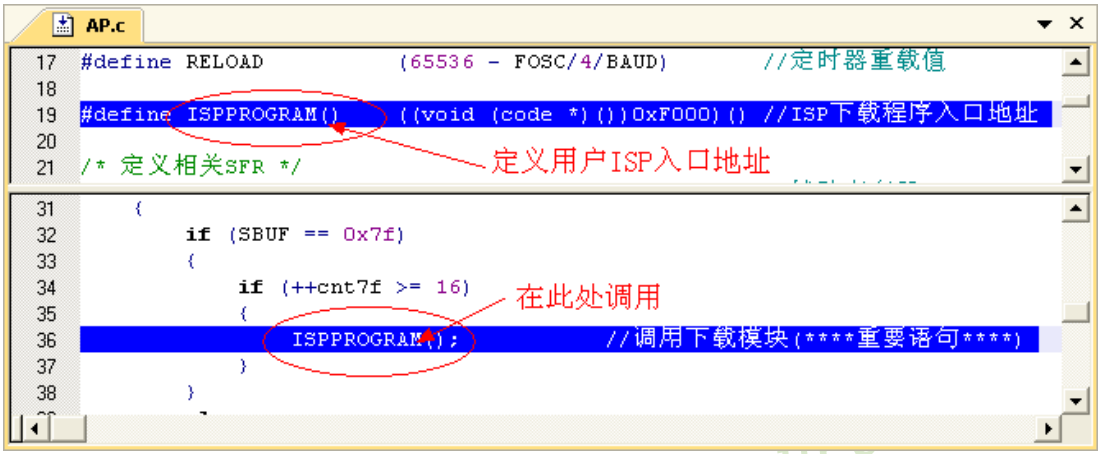
```
START:
        MOV     R0,     #7FH                    ;清RAM
        CLR     A
        MOV     @R0,    A
        DJNZ    R0,     $-1
        MOV     SP,     #7FH                    ;初始化SP

        MOV     SCON,   #50H                    ;设置串口模式(8位数据位,波特率可变,无校验位)
        MOV     AUXR,   #40H                    ;T1工作于1T模式
        MOV     TH1,    #HIGH UARTBAUD          ;波特率设置
        MOV     TL1,    #LOW UARTBAUD           ;
        SETB    TR1                             ;启动定时器1
        SETB    ES                              ;使能串口中断
        SETB    EA                              ;开中断总开关

MAIN:
        INC     P0                              ;用户示例代码
        SJMP    MAIN
;----------------------------------------------
        END
```

对于汇编语言编写的用户代码需要注意一点：位于0000H的复位入口地址处的指令必须是一个长跳转语句（类似LJMP START）。在用户代码中，需要设置好串口，并在满足下载条件时，跳转到ISPPROGRAM（即F000H的绝对地址），以实现代码更新。

对于汇编语言编写的用户代码，我们可以使用如下图的方法进行调用：



对于C语言编写的用户代码，我们可以使用如下图的方法进行调用：



四．上位机应用程序说明

上位机应用程序请用户在STC官方网站www.STCMCU.com下载。

上位机演示程序是基于MFC的对话框项目，对于串口的访问是直接调用Windows的API函数，而没有使用串口控件，从而省去的控件的注册以及系统版本不兼容的诸多问题。界面较简单，只是为这一功能的实现提供了一个框架，其他的功能及要求均还可以往上面添加。

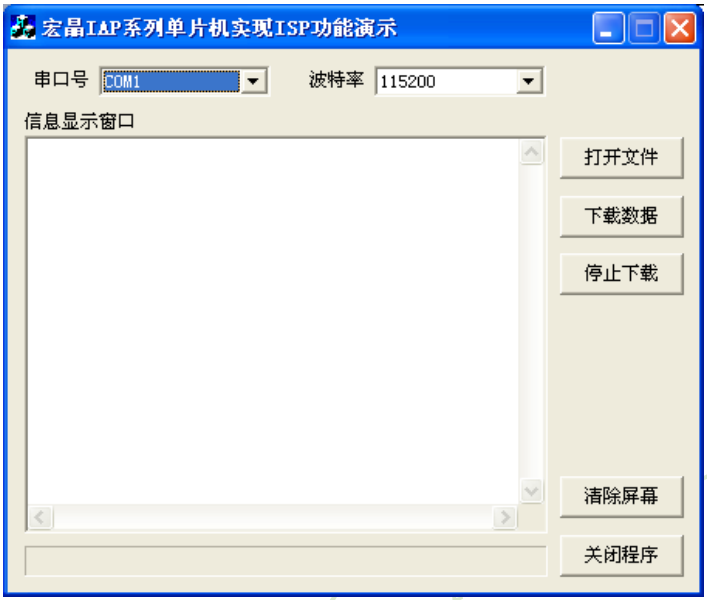
上位机程序的核心模块是基于类CISPDlg的一个友元函数“UINT Download(LPVOID pParam)；”。

```
// Construction
public:
    CISPDlg(CWnd* pParent = NULL);  // standard constructor

    friend UINT Download(LPVOID pParam);
```

此函数负责与下位机通讯，发送各种通讯命令来完成对用户程序的更新。用户可以根据各自不同的需求增加命令。

. 上位机应用程序的使用方法

1、打开上位机界面，如下图



2、选择串口号，设置与下位机相同的串口波特率

3、打开要下载的源数据文件，Bin或者Intel hex格式均可以

4、点击“下载数据”按钮即可开始下载数据

. 下位机固件程序的使用方法

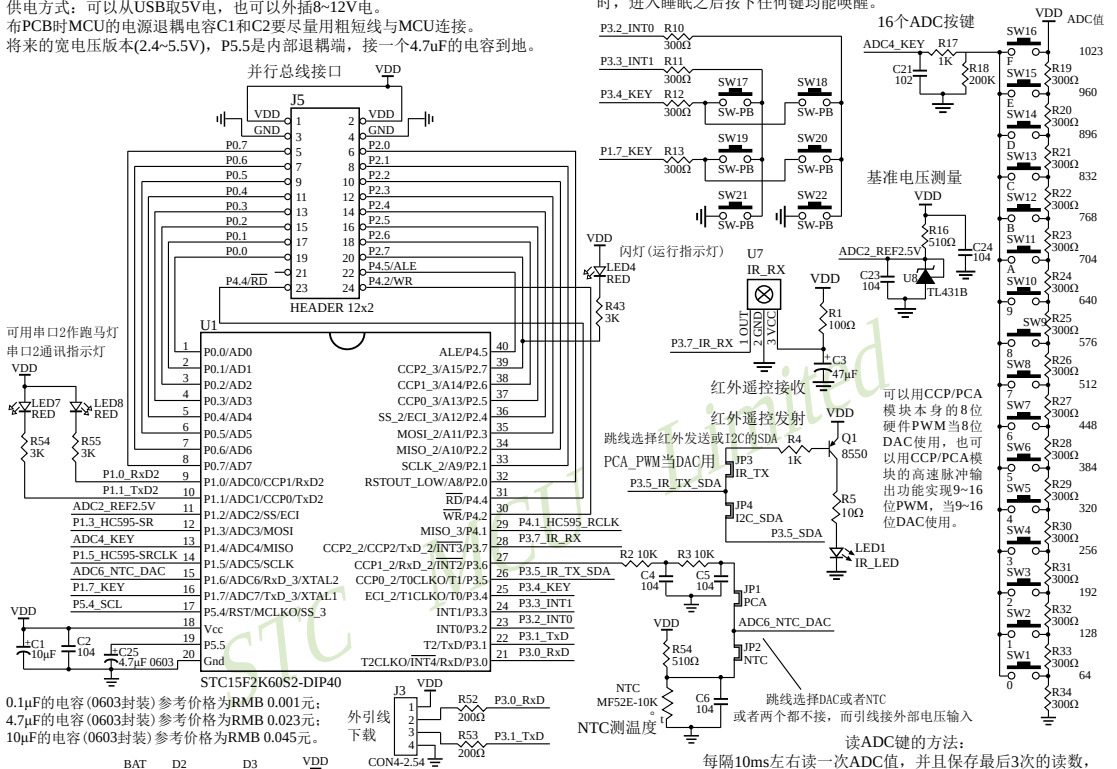
下位机的目标文件有两个：“IAPISP.hex”和“AP.hex”，对于一块新的单片机，第一次必须使用宏晶科技的ISP下载工具将“IAPISP.hex”写入到芯片内。附件中的“AP.hex”是用户程序模板，当满足下载条件时（模板代码中的下载条件是连续就收到16个7FH的数据），用户将程序跳转到用户ISP入口地址（F000H）处，即可实现代码更新。

附录K：STC15F2K60S2系列学习板电路示意图

说明：

IAP15F/15L2K61S2、STC15F/15L2K08S2可以仿真，但STC15F/15L2K08S2更便宜。
SW23是下载断电按钮，下载时按一下再释放就可以冷启动。
供电方式：可以从USB取5V电，也可以外插8~12V电。
布PCB时MCU的电源退耦电容C1和C2要尽量用粗短线与MCU连接。
将来的宽电压版本(2.4~5.5V)，P5.5是内部退耦端，接一个4.7uF的电容到地。

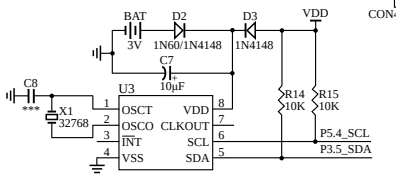
6个可唤醒按键，可作红外遥控发射或接收学习，或别的功能。
睡眠时，先将P3.4_KEY和P1.7_KEY输出低，P3.2_INT0和P3.3_INT1输出高，并且在P3.2_INT0和P3.3_INT1都是高电平时，进入睡眠之后按下任何键均能唤醒。



0.1μF的电容(0603封装)参考价格为RMB 0.001元；
4.7μF的电容(0603封装)参考价格为RMB 0.023元；
10μF的电容(0603封装)参考价格为RMB 0.045元。

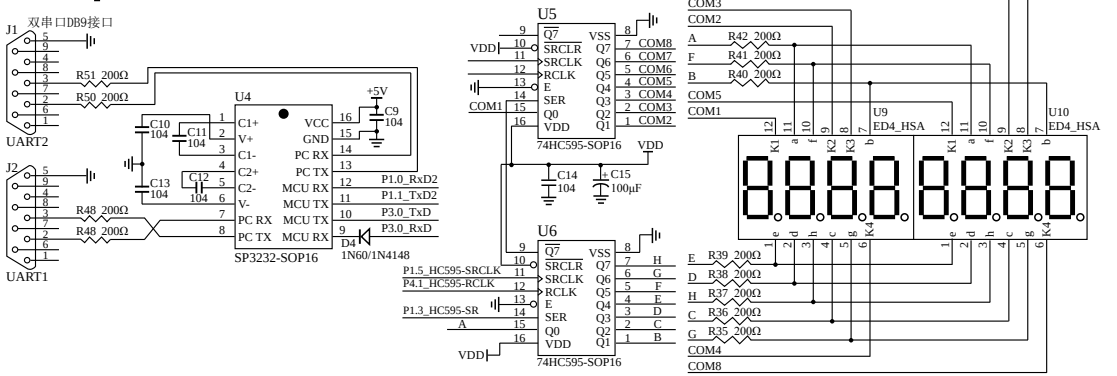
读ADC键的方法：

每隔10ms左右读一次ADC值，并且保存最后3次的读数，其变化比较小时再判断键。判断键有效时，允许一定的偏差，比如±16个字的偏差。

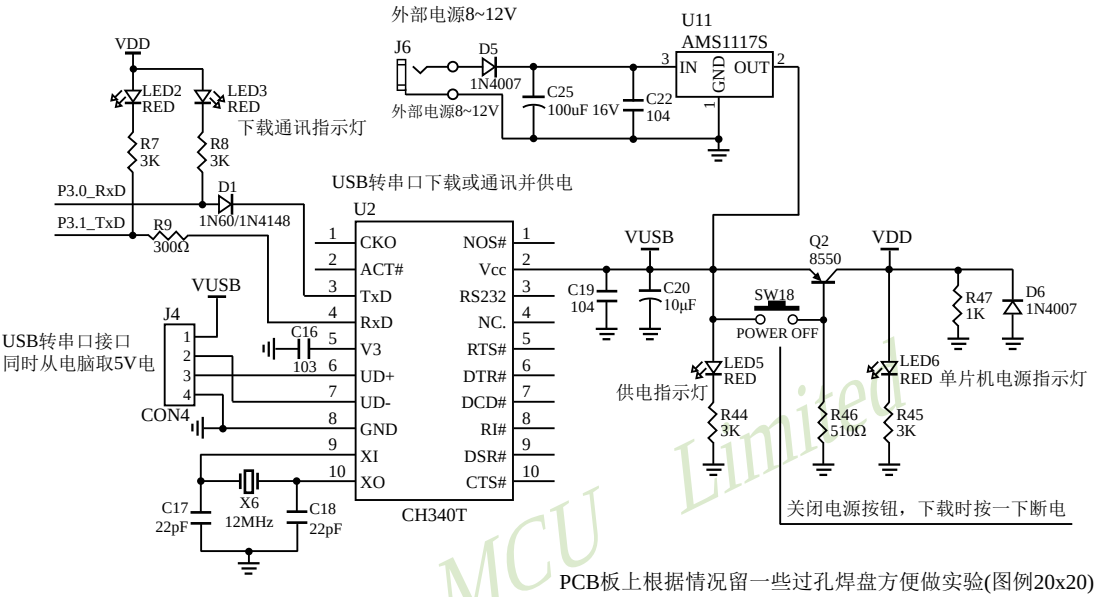


两片74HC595驱动8个数码管　74HC595(SOP16封装)参考价格为RMB 0.2元。
数码管使用共阴极比较好，因为595拉低能力比较强

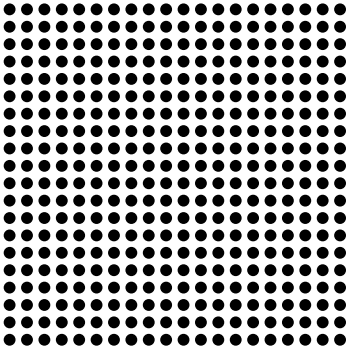
(共阴极/共阳极均可)



STC15F2K60S2系列学习板电路示意图续

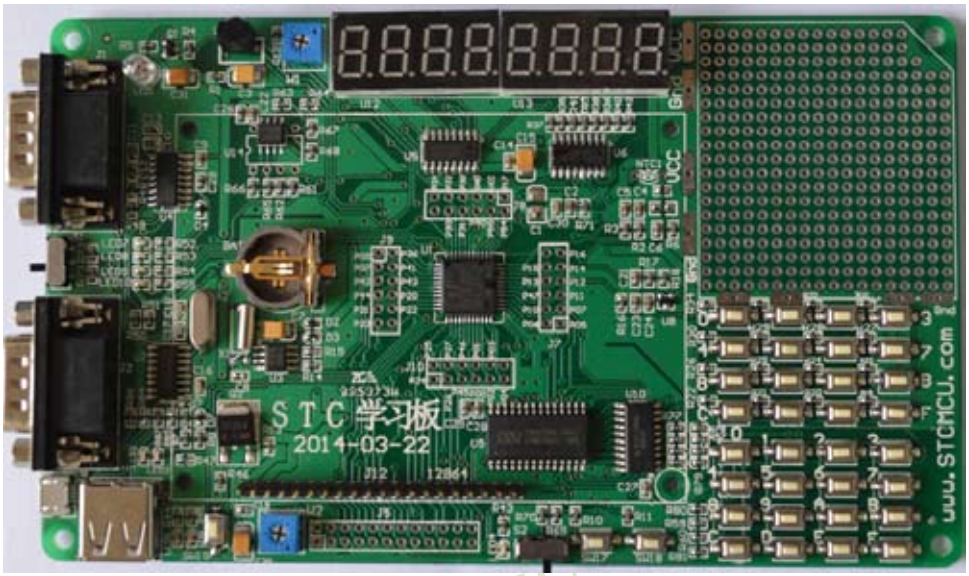


PCB板上根据情况留一些过孔焊盘方便做实验(图例20x20)

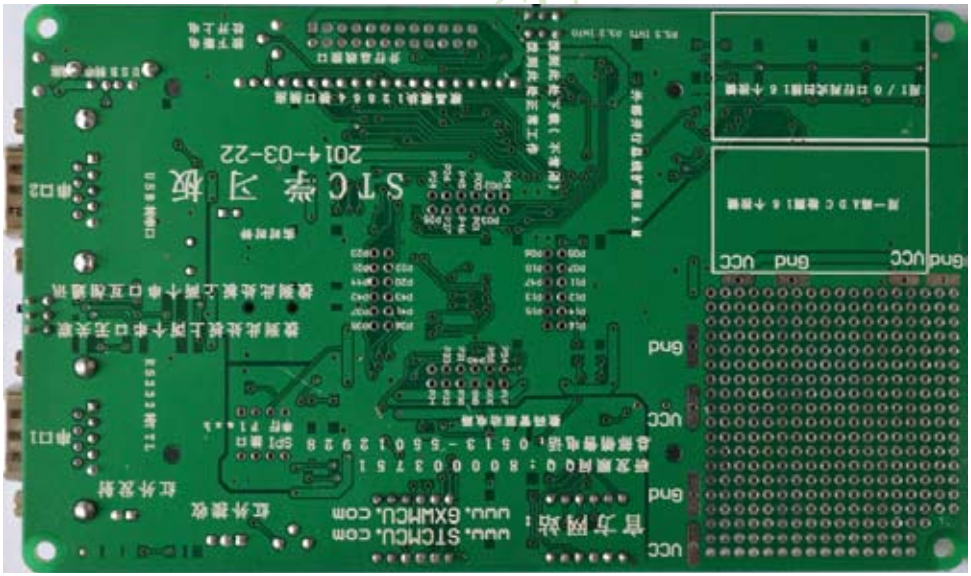


STC MCU Limited

可仿真的STC15系列学习板4的实物图



STC学习板4实物图正面



STC学习板4实物图反面

可仿真的STC15系列学习板4的运行测试程序下载地址如下：

http://www.stcmcu.com/STC-Study/STC-%E5%BC%80%E5%8F%91%E6%9D%BF4-%E7%A8%8B%E5%BA%8F-C%E8%AF%AD%E8%A8%80-V1.RAR

附录L：逻辑代数的基础

——无微机原理的用户请从本章开始学习

这一章主要讲述的内容有：①在数字设备中进行算术运算的基本知识——数制和编码；②数字电路中一些常用逻辑运算及其图形符号。它们是学习单片机这门课程的基础。对于没有微机原理基础的用户和同学，请从这章开始学习。

L.1 数制与编码

数制是人们利用符号进行计数的科学方法。数制有很多种，常用的数制有：二进制，十进制和十六进制。

进位计数制是把数划分为不同的位数，逐位累加，加到一定数量之后，再从零开始，同时向高位进位。进位计数制有三个要素：数码符号、进位规律和计数基数。下表是各常用数制的总体介绍。

常用的数制	表示符号	数码符号	进制规律	计数基数
二进制	B	0、1	逢二进一	2
十进制	D	0、1、2、3、4、5、6、7、8、9	逢十进一	10
十六进制	H	0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F	逢十六进一	16

我们日常生活中计数一般采用十进制。计算机中采用的是二进制，因为二进制具有运算简单，易实现且可靠，为逻辑设计提供了有利的途径、节省设备等优点。为区别于其它进制数，二进制数的书写通常在数的右下方注上基数2，或加后面加B表示。二进制数中每一位仅有0和1两个可能的数码，所以计数基数为2。二进制数的加法和乘法运算如下：

0+0=0　　0+1=1+0=1　　1+1=10

0×0=0　　0×1=1×0=0　　1×1=1

由于二进制数在使用中位数太长,不容易记忆，为了便于描述，又常用十六进制作为二进制的缩写。十六进制通常在表示时用尾部标志H或下标16以示区别。

L.1.1 数制转换

现在我们来介绍这些常用数制之间的转换。

一：二进制 — 十进制转换

方法：将二进制数按权(如下式)展开，然后将各项的数值按十进制数相加，就得到相应的等值十进制数。

例如：N=(1101.101)B，那么N所对应的十进制数时多少呢？

按权展开$N=1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}=8+4+0+1+0.5+0+0.125=(13.625)D$

二：十进制 — 二进制转换

方法：分两部分进行即整数部分和小数部分。

①整数部分转换(基数除法)：

★ 把我们要转换的数除以二进制的基数(二进制的基数为2)，把余数作为二进制的最低位；

★ 把上一次得的商在除以二进制基数(即2)，把余数作为二进制的次低位；

★ 继续上一步,直到最后的商为零,这时的余数就是二进制的最高位.

②小数部分转换(基数乘法)：

★把要转换数的小数部分乘以二进制的基数(二进制的基数为2)，把得到的整数部分作为二进制小数部分的最高位；

★把上一步得的小数部分再乘以二进制的基数(即2)，把整数部分作为二进制小数部分的次高位；

★继续上一步，直到小数部分变成零为止。或者达到预定的要求也可以。

例如：将$(213.8125)_{10}$化为二进制数可按如下进行：

先化整数部分：

```
2 | 213 ------ 余数=1=k0
2 | 106 ------ 余数=0=k1
2 |  53 ------ 余数=1=k2
2 |  26 ------ 余数=0=k3
2 |  13 ------ 余数=1=k4
2 |   6 ------ 余数=0=k5
2 |   3 ------ 余数=1=k6
2 |   1 ------ 余数=1=k7
    0
```

于是整数部分$(213)_{10}=(11010101)_2$

再化小数部分：

```
  0.8125
×      2
 1.6250 ------ 整数部分=1=k-1

  0.6250
×      2
 1.2500 ------ 整数部分=1=k-2

  0.2500
×      2
 0.5000 ------ 整数部分=0=k-3

  0.5000
×      2
 1.0000 ------ 整数部分=1=k-4
```

于是小数部分$(0.8125)_{10}=(0.1101)_2$

综上所述，十进制数213.8125=$(11010101.1101)_2$=(11010101.1101)B

STC MCU Limited

三：二进制 — 十六进制转换

方法：二进制和十六进制之间满足2^4的关系，因此把要转换的二进制从低位到高位每4位一组，高位不足时在有效位前面添“0”，然后把每组二进制数转换成十六进制即可。

例如，将(010111011110.11010010)B转换为十六进制数：

(0101 1101 1110 . 1101 0010)B
　　↓　　↓　　↓　　↓　　↓
=(5　　D　　E　　B　　2)H

于是，(010111011110.11010010)B=(5DE.B2)H

四：十六进制 — 二进制转换

方法：十六进制转换为二进制时，把上面二进制转换十六进制的过程逆过来，即转换时只需将十六进制的每一位用等值的4位二进制代替就行了。

例如：将(C1B.C6)H转换为二进制数：

(C　　1　　B .　C　　6)H
　↓　　↓　　↓　　↓　　↓
=(1100　0001　1011　1100　0110)B

于是，(C1B.C6)H=(110000011011.11000110)B

五：十六进制 — 十进制转换

方法：将十六进制数按权(如下式)展开，然后将各项的数值按十进制数相加，就得到相应的等值十进制数。

例如：N=(2A.7F)H，那么N所对应的十进制数时多少呢？

按权展开$N=2\times16^1+10\times16^0+7\times16^{-1}+15\times16^{-2}=32+10+0.4375+0.05859375=(42.49609375)D$

于是，(2A.7F)H =(42.49609375)D

六：十进制 — 十六进制转换

方法：将十进制数转换为十六进制数时，可以先将十进制数转换为二进制数，然后再将得到的二进制数转换为等值的十六进制数。

STC MCU Limited

L.1.2 原码、反码及补码

在生活中,数有正负之分,在计算机中是怎样表示数的正负符号呢?

在生活中表示数的时候一般都是把正数前面加一个“+”，负数前面加一个“-”，但是计算机是不认识这些的，通常在二进制数前面增加一位符号位。符号位为“0”表示“+”，符号位为“1”表示“-”。这种形式的二进制数称为原码。如果原码为正数，则原码的反码和补码都与原码相同。如果原码为负数，则将原码(除符号位外)按位取反，所得的新二进制数称为原码的反码，反码加1为其补码。

原码、反码、补码这三种形式的总结如下表所示：

	真值	原码	反码	补码
正数	+N	0N	0N	0N
负数	-N	1N	$(2^n-1)+N$	2^n+N

例1：求+18和-18八位原码、反码、补码形式。

真值	原码	反码	补码
+18	00010010	00010010	00010010
-18	10010010	11101101	11101110

L.1.3 常用编码

指定某一组二进制数去代表某一指定的信息，就称为编码。

一：十进制编码

用二进制码表示的十进制数，称为十进制编码。它具有二进制的形式，还具有十进制的特点它可作为人们与数字系统的联系的一种间表示。十进制编码有很多种，最常用的一种是BCD码，又称8421码。

下面我们用表列出几种常见的十进制编码:

编码 / 种类 / 十进制数	8421码（BCD码）	余3码	2421码	5211码	7321码
0	0000	0011	0000	0000	0000
1	0001	0100	0001	0001	0001
2	0010	0101	0010	0100	0010
3	0011	0110	0011	0101	0011
4	0100	0111	0100	0111	0101
5	0101	1000	1011	1000	0110
6	0110	1001	1100	1001	0111
7	0111	1010	1101	1100	1000
8	1000	1011	1110	1101	1001
9	1001	1100	1111	1111	1010
权	8421		2421	5211	7321

STC MCU Limited

十进制编码分为有权和无权编码。有权编码码是指每一位十进制数符均用一组四位二进制码来表示，而且二进制码的每一位都有固定权值。无权编码码是指二进制码中每一位都没有固定的权值。 上表中8421码(即BCD码)、2421码、5211码、7321码都是有权编码，而余3码是无权编码。

二： 奇偶校验码

在数据的存取、运算和传送过程中，难免会发生错误，把“1”错成“0”或把“0”错成“1”。奇偶校验码是一种能检验这种错误的代码。它分为两部分；信息位和奇偶校验位。有奇数个“1”称为奇校验，有偶数个“1”则称为偶校验。

STC MCU Limited

L.2 几种常用的逻辑运算及其图形符号

逻辑代数中常用的运算有：与(AND)、或(OR)、非(NOT)、与非(NAND)、或非(NOR)、与或非(AND-NOR)、异或(EXCLUSIVE OR)、同或(EXCLUSIVE NOR)等。其中与(AND)、或(OR)、非(NOT)运算时三种最基本的运算。

一：与运算及与门

与运算：决定事件结果的全部条件同时具备时，事件才发生。
逻辑变量A和B进行与运算时可写成：Y=A • B

真值表		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

与门：实行与逻辑运算的单元电路。
与门图形符号：

二：或运算及或门

或运算：决定事件结果的各条件中只要有任何一个满足，事件就会发生。
逻辑变量A和B进行或运算时可写成：Y=A+B

真值表		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

或门：实行或逻辑运算的单元电路。
或门图形符号：

三：非运算及非门

非运算：条件具备时，事件不会发生；条件不具备时，事件才会发生。
逻辑变量A进行非运算时可写成：Y=A′

真值表	
A	Y
0	1
1	0

非门：实行非逻辑运算的单元电路。
非门图形符号：

STC MCU Limited

四：与非运算及与非图形符号

与非运算：先进行与运算，然后将结果求反，最后得到的即为与非运算结果。
逻辑变量A和B进行与非运算时可写成：Y=(A・B)′

真值表		
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

与非图形符号：

五：或非运算及或非图形符号

或非运算：先进行或运算，然后将结果求反，最后得到的即为或非运算结果。
逻辑变量A和B进行或非运算时可写成：Y=(A+B)′

真值表		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

或非图形符号：

六：与或非运算及与或非图形符号

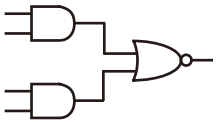
与或非运算：在与或非逻辑运算中有4个逻辑变量A、B、C、D。假设A和B为一组，C和D为一组，A、B之间以及C、D之间都是与的关系，只要A、B或C、D任何一组同时为1，输出Y就是0。只有当每一组输入都不全是1时，输出Y才是1。
逻辑变量A和B进行或非运算时可写成：Y=(A・B+C・D)′

真值表				
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1

STC MCU Limited

1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

(接上表)

与或非图形符号： 

七：异或运算及异或图形符号

异或运算：当A、B不同时，输出Y为1；而当A、B相同时，输出Y为0。
逻辑变量A和B进行异或运算时可写成：$Y = A \oplus B = (A \cdot B') + (A' \cdot B)$

真值表		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

异或图形符号：

八：同或运算及同或图形符号

同或运算：当A、B不同时，输出Y为0；而当A、B相同时，输出Y为1。
逻辑变量A和B进行同或运算时可写成：$Y = A \odot B = (A \cdot B) + (A' \cdot B')$

真值表		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

同或图形符号：

STC MCU Limited

附录M：STC对单片机相关学科群部分课程改革呼吁

工信部一位对推动中国单片机教学改革有兴趣的朋友约我对这方面提一点看法，与高校及企业合作多年的经验，我确实也深有感触。

STC自成立起就一直专注于国内流行的8051单片机的研发工作，和各高校以及企业保持着密切的联系和紧密的合作。根据我们STC与高校及企业合作多年的经验，我们深刻感受到当前大学里单片机课程教育存在的弊端，主要是开设的时间和顺序以及完整性，不需要增加教学时间，还可适当减少课时，当前单片机课程教育的改革迫在眉睫。对此我们提出以下建议。

单片机相关课程教学之最小系统课程：

《C语言程序设计》	大学一年级第一学期
《数字电路基础》	大学一年级第一学期
《微机原理及单片机应用》	大学一年级第二学期
《数据结构》	大学一年级第二学期
《实时操作系统》	大学二年级第一学期
《模拟电路》	大学二年级第一学期

STC MCU Limited

首先，我们建议《C语言程序设计》课程必须保证在大学一年级第一学期开设，很多高校已这样做了，但还有部分学校滞后了，我的观念晚学不如早学，工科非工科都应将此门课作为进入信息时代的计算机扫盲课（word/excel/打字就由学生自学吧），学好了《C语言程序设计》课程就打好了单片机学习中的部分软件基础。相对于用汇编语言开发程序，C语言也将程序开发难度降低了很多，对开发人员来说，效率得到极大的提高，打个不恰当的比方，

大专生就相当于本科生的能力了，本科生就相当于研究生的能力了。

其次，我们强烈建议将《数字电路》课程提前放在大学一年级第一学期开设，《数字电路》能帮助学生理解微机原理和基本的数字系统，并能设计一些简单的硬件电路系统，是单片机学习的硬件基础。

再次，就是要学习《微机原理及单片机应用》了，我们强烈建议将《微机原理及单片机应用》课程提前到大学一年级第二学期开设。许多学校的相关工科专业都是在大学三年级第二学期才开设这门课程。由于接触单片机的时间晚，学生根本没时间认真学习和研究单片机，更没时间应用所学的知识开发一些单片机项目。在许多学生只学到一些单片机皮毛的情况下，他们便匆匆步入社会进入了相关行业。这种情况往往导致很多学生不能胜任自己的岗位，工作起来很吃力，而且极容易被淘汰。这就有了“学生抱怨就业难，企业却反映招不到人才”的怪现象。如果将《微机原理及单片机应用》课程提前至大学一年级第二学期开设，学生及早接触单片机的话，那么对单片机有兴趣或以后想从事相关行业的同学在意识到单片机的重要性后，仍然有充足的时间去研究单片机，从而将单片机系统设计搞熟。这样在他们毕业以后进入相关行

业及企业工作就会很快能胜任。单片机课程不是一门纯理论性的课程，更多的需要学生去动手实践和开发，所以学习单片机以及开发单片机项目能提高学生的创新能力，如果学得太晚就很难学以致用。其实很多工科的专业课程都如此，很多学生在学习完这些专业课程后之所以云里雾里、不知所云，就是因为没有花时间或没有足够的时间去研究和实践应用所学的内容。如果这些动手实践型的专业课程开设的太晚，学生在学习完之后根本没有时间去动手应用的话，那么这些课程就会形同虚设，对学生以后的就业将起不到应有的帮助作用。

另外，为了使学生在开发程序时，能得心应手，我们建议在大学一年级第二学期应再开设《数据结构》课程，提高开发效率，降低开发难度，同时如果《C语言程序设计》没学好，通过对《数据结构》的学习，也可算对《C语言程序设计》的复习，为单片机开发打好软件基础。还是那个不恰当的比方，有《数据结构》作为基础比没有《数据结构》作为基础的，大专生就相当于本科生的能力了，本科生就相当于研究生的能力了。

最后，我们建议在大学二年级第一学期开设《实时操作系统》和《模拟电路》等后续课程。这些课程是对前面所学课程的补充和提高，如果《数据结构》没学好，学习《实时操作系统》也可算对《数据结构》的复习，为单片机开发打好软件基础。基于实时操作系统的单片机开发，对开发者的能力要求可大幅降低，能较容易开发出大型、复杂的单片机项目。还是那个不恰当的比方，有《实时操作系统》作为基础比没有《实时操作系统》作为基础的，大专生就相当于本科生的能力了，本科生就相当于研究生的能力了。

有了以上基础，相信大学生创新竞赛会更有价值！

乘风破浪会有时，直挂云帆济沧海。

相信单片机课程教育的改革会给当前的单片机教育和科技创新开创一片新的天地，相信各高校一定能培养出更多、更专业、更符合社会需求的优秀人才。

STC MCU Limited

附录N：STC对单片机课程教育改革的贡献

当前绝大部分高校都以8051单片机作为单片机课程教育的基础，但普通的8051单片机诞生与上世纪70年代，不可避免地面临着落伍的危险。为此，STC宏晶科技对8051单片机进行了全面的技术升级与创新，对当前单片机课程教育的改革有巨大的贡献。

宏晶科技STC单片机设计公司是全球最大的8051单片机设计公司，致力于开发设计1T增强型8051单片机，速度平均比普通8051快7~12倍，指令代码完全兼容普通8051单片机。STC单片机全部采用Flash技术(可反复编程10万次以上)和ISP/IAP(在系统可编程/在应用可编程)技术，并对传统8051进行了全面提速，指令最快提高了24倍。另外，STC针对抗干扰进行了专门设计，针对加密进行了特别加密设计，如宏晶STC15系列超强抗干扰，现无法解密。其次，STC单片机大幅提高了集成度，如集成了A/D、CCP/PCA/PWM(PWM还可当D/A使用)、高速同步串行通信端口SPI、高速异步串行通信端口UART(如宏晶STC15F2K60S2系列集成了两个串行口，分时复用可当5组串口使用)、定时器(宏晶STC15F2K60S2系列最多可达到6个定时器)、看门狗、高可靠复位电路(可彻底省掉外部复位)、内部高精准时钟(-40℃ ~ +85℃之间最大只有±1%的温飘，可彻底省掉外部昂贵的晶振)、大容量SRAM(如宏晶STC15F2K60S2系列集成了2K字节的SRAM)、大容量EEPROM、大容量Flash程序存储器等。STC单片机几乎包含了数据采集和控制中所需要的所有单元模块，可称得上是一个真正的片上系统(SysTem Chip或SysTem on Chip, 简写为STC, 这是宏晶科技STC名称的由来)。可以说，STC单片机来源于普通8051单片机，却又高于普通8051单片机。如果学生或单片机爱好者以STC单片机作为学习的工具，将会对8051单片机有个更全面更透彻的掌握。

STC单片机除对普通8051单片机的技术技术进行升级外，还对普通8051单片机一些复杂难懂的知识点进行了简化。例如，对初学者而言，定时器T0/T1的四种工作模式只需学习其中的模式0(16位自动重装载模式)，定时器0的模式3(不可屏蔽中断的16位自动重装载模式)还可作实时操作系统节拍定时器，定时器2也只需学习一种模式。另外，串行口的波特率计算公式也比普通8051的计算公式简单的多。例如，我们用定时器2作为串行口的波特率发生器，则串行口的波特率计数公式为：

串行口的波特率=(定时器T2的溢出率)/4 ；　　注意：此波特率与SMOD无关

……对教学而言，这些简化减少了教学的课时，从而大大减轻了教学压力。

STC公司设计生产的一些辅助工具，如STC-ISP下载编程工具，能够更好地帮助学生和单片机爱好者更容易地学习和理解STC单片机。对于单片机学习者而言，波特率的计算、定时器的计算、头文件的编写等一直是重点和难点，很多学习者在学习完单片机后不知道如何计算波特率、定时器的参数等等。最新的STC15系列ISP下载编程工具集成了波特率计算器、定时器计算器、软件延时器、头文件等有用的工具，所以使用STC单片机的用户再也不用担心不会计算波特率、定时器等参数了。

从宏晶科技STC单片机设计公司多年的发展来看，使用STC单片机的用户越来越多，可见STC对高校课程教育改革乃至对社会贡献的巨大。

附录O：STC推荐的单片机教材

O.1 两本基于可仿真的STC15F2K60S2系列单片机的本科教材

一、《单片微型计算机原理及接口技术》，作者陈桂友，高等教育出版社出版

《单片微型计算机原理及接口技术》，山东大学陈桂友教授主编，姚永平、王威康主审，由原教育部副部长吴启迪教授和教育部高等学校自动化专业分委员会主任、中国工程院院士清华大学吴澄教授共同作序，高等教育出版社出版，于2012年4月出版，受到热烈欢迎，至今已第三次印刷，得到了国内众多高校的教师普遍认可，计划2014年推出第二版。

此教材以可仿真的STC15F2K60S2系列单片机为背景，从介绍微型计算机的基本结构和工作原理入手，介绍单片微型计算机（单片机）的构成、各个模块的工作过程、接口原理、应用电路设计、汇编语言和C语言程序设计，精选应用实例，强化单片机技术的实践性与应用性，内容尽可能地选择了目前实际工程中常用的新技术、新器件，力图达到学以致用的根本目的。

全书共分12章，第1章简要介绍微型计算机的发展历史及应用；第2章介绍微型计算机的基础知识，包括数制编码、微型计算机的常见电路、常见技术术语等；第3章介绍计算机系统的组成与工作原理，介绍模型机的构成及工作过程，并介绍单片机的内部结构及典型系统构成；第4章介绍单片机的指令系统及汇编语言程序设计，介绍单片机程序仿真调试和下载的方法；第5章介绍单片机的C语言程序设计与调试，介绍C语言与单片机汇编语言之间的联系；第6章介绍中断的概念和单片机的中断系统；第7章介绍定时器计数器与可编程计数器阵列；第8章介绍数据通信技术，主要介绍常用的并行接口和串行接口工作原理、接口方法以及常用的数据接口芯片及其使用实例；第9章是模数转换器与数模转换器，分别介绍两种转换器的原理和典型芯片的应用；第10章介绍人机交互接口设计，人机交互接口是单片机应用系统必不可少的接口应用；第11章介绍单片机系统的看门狗技术、时钟选择及省电方式的原理和技术；第12章介绍应用系统的设计实例，从硬件和软件两个方面介绍应用系统的设计。每章都有配套的习题，所举例程均经调试通过，很多程序均来自科研和实际应用系统。为了便于学习，开发了与教材配套的综合教学实验平台，该平台提供了20余种实验供学生选用学习，也为善于思考、乐于动手实践的学生提供了自学习实验手段。

本书深入浅出，层次分明，实例丰富，通俗易懂，突出实用，可操作性强，特别适合于作为普通高校计算机类、电子类、电气自动化及机械专业的教学用书。还可作为高职高专以及培训班的教材使用。同时，也可作为从事单片机应用领域的工程技术人员的参考书。

网上订购：当当网、京东商城、亚马逊等

二、《单片微机原理及接口技术》——基于可仿真的STC15系列单片机，作者丁向荣

《单片微机原理与接口技术(ASM+C) -----基于STC15系列单片机》 ，作者丁向荣，本教材于2012年9月出版，于2013年8月第二次印刷，使用本教材的大学有中国矿业大学、深圳大学、河海大学、九江学院、苏州大学、北方理工大学、东北石油大学、北方工业大学等。

此教材选用可在线仿真、在线编程、内置复位电路与时钟电路的STC15系列单片机；将微机原理与单片机技术有机结合，汇编语言与C语言“双语言”对照编程； 精选应用实例，强化单片机技术的实践性与应用性。

在全国各大书店和当当网、京东商城、亚马逊等网店有售

该书内容简介：

STC15系列增强型8051单片机集成了上电复位电路与高精准R/C振荡器，给单片机芯片加上电源就可跑程序；集成了大容量的程序存储器、数据存储器以及EEPRM，集成了A/D、PWM、SPI等高功能接口部件，可大大地简化单片机应用系统的外围电路，促使单片机应用系统的设计更加简捷，系统性能更加高效、可靠。本教材以STC15F2K60S2单片机为主线，系统地介绍了STC15F2K60S2单片机的硬件结构、指令系统与应用编程，系统地介绍了单片机应用系统的开发流程与接口设计，同时，提出多种实践模式：Keil C集成开发环境、Proteus仿真软件以及实物运行开发环境，使得单片机的学习与应用变得更简单、更清晰。

本书可作为普通高校计算机类、电子信息类、电气自动化与机电一体化等专业的教学用书，基础较好的高职高专也可选用本书。此外，可作为电子设计竞赛、电子设计工程师考证的培训教材。也是传统8051单片机应用工程师升级转型的重要参考书籍。

O.2 一本基于可仿真的STC15系列单片机的高职高专教材

《增强型单片机8051单片机原理与系统开发（C51版）》，作者丁向荣，于2013年9月清华大学出版。本教材可作为高职或应用本科计算机类、电子信息类、电气自动化与机电一体化等专业的教学用书。此外，本书可作为电子设计竞赛、电子设计工程师考证的培训教材，也是传统8051单片机应用工程师升级转型的重要参考书。

此教材基于可在线仿真、在线编程、内置复位电路与时钟电路的STC15系列单片机，工学结合，采用任务驱动模式组织教材内容，将教学内容嵌入到一个个单片机应用系统中，学习单片机就是在做单片机应用系统，可实施“教、学、做”一体化教学模式，能有效提高单片机应用实践能力与编程能力。

本教材在当当网、京东商城、亚马逊等网店有售。

STC15F2K60S2系列增强型8051单片机集成了上电复位电路与高精准RC振荡器，给单片机芯片加上电源就可以运行程序；集成了大容量的程序存储器、数据存储器以及E2PROM，集成了A/D、PWM、SPI等高功能接口部件，可大大地简化单片机应用系统的外围电路，促使单片机应用系统的设计更加简便、快捷，系统性能更加高效、可靠。

STC15F2K60S2单片机的可仿真技术是STC系列单片机的一大创举，它可自定义为仿真芯片或目标应用芯片，仿真时无需增加任何电路，使得单片机仿真变得简单而实用。

本教材按照“教、学、做”一体化教学模式组织教学内容，分基础篇与应用篇，共17个项目，42个任务，兼顾少学时与多学时教学体系。

本书可作为高职或应用本科计算机类、电子信息类、电气自动化与机电一体化等专业的教学用书。此外，本书可作为电子设计竞赛、电子设计工程师考证的培训教材，也是传统8051单片机应用工程师升级转型的重要参考书。

附录P：STC公司简介及招聘

P.1 STC公司简介

STC是全球最大的8051单片机设计公司，STC是SysTem Chip(系统芯片)的缩写，因性能出众，领导着行业的发展方向，被用户评为8051单片机全球第一品牌，51中的战斗机，创始人姚永平给他取了三个中文名：香港宏晶科技有限公司、深圳宏晶科技有限公司、南通国芯微电子有限公司。

南通国芯微电子享受南通市的高度支持，作为南通/江苏/中国的高科技龙头，销售大户和纳税大户，南通科技园给了2000平方米的写字楼欢迎STC全球运营总部落户南通，市委书记亲自前往指导工作。现使用面积达8000平方米的STC全球运营总部大楼在南通已建成，全面装修后即将投入使用。STC经过十余年的高速发展, 全球规模已达150余人，研发团队30余人（以留美海归/恢复高考后的第一批大学生/台湾工研院等背景的资深研发人员为主，曾经主导过美国SST的8051设计，主导过台湾华邦的8051设计，参与过日本东芝的MCU设计）。

国内办事处有：南通/深圳/广州/东莞/上海/南京/北京

台湾办事处有：台北/新竹

即将在全球建成完整的分销网络

STC南通国芯微电子现具备 0.35uM/0.18uM/0.13uM/90nM 的高阶数模混合集成电路设计技术，现设计的芯片在TSMC上海流片生产(全球最大的晶元代工厂,同时在给Ti/Freescale/NXP/Atmel服务），在南通富士通封装（大陆本土最大的封装厂，同时在给Atmel…服务）。

STC MCU Limited

P.2 STC公司招聘

STC公司（北京/上海/广州/深圳 分公司）现面向全社会招聘销售工程和销售助理，具体要求如下：

（1）熟练掌握单片机的外部中断、定时器、串行口、ADC等的使用；

（2）熟悉STC15F2K60S2系列单片机，会使用IAP15F2K61S2单片机做仿真；

（3）参加过单片机等电子设计大赛的优先。

欢迎有志于从事单片机事业的人才加入，联系邮箱：STC13922805190@qq.com

附录Q：每日更新内容的备忘录

2014-7-2更新内容：

（1）STC15W100型号单片机价格更改为RMB 1.05元；
（2）当串口1、串口3、串口4和串口2的波特率相同时，串口1、串口3、串口4和串口2可共享定时器T2作波特率发生器
（3）更新4.22节“用STC MCU的I/O口直接驱动段码LCD的原理及扫描程序”

2014-6-9更新内容：

（1）更新了彩色宣传资料；
（2）仅STC15L104W系列建议用STC15W104系列代替
（3）脱机/联机下载工具U7板上的三极管T12若损坏用SS8050更换

2014-5-24更新内容：

（1）将彩色宣传册中的STC15F101W系列的宣传册替换为STC15W104系列宣传册
（2）更正了附录中公司招聘中的错误
（3）将正文中各系列简介中的选型表格式制作为与彩色宣传册中的选型表格式一致

2014-5-21更新内容：

（1）修正6.4节“在KEIL C中如何声明中断函数”中Timer2与PWM占用中断号重复的错误；
（2）STC15W408AS系列单片机新增封装为QFN28（5mm x 5mm）的单片机，新增的封装为QFN28单片机的价格与封装为SKDIP28的价格相同；
（3）增加QFN28封装尺寸图；
（4）STC15F/L2K60S2系列单片机的D版本开始供货；
（5）将“以15L开头的芯片如需进入“掉电模式”，在进入“掉电模式”前必须启动掉电唤醒定时器<3uA>,不超过3秒要唤醒一次”中的“不超过3秒要唤醒一次”修改为“不超过1秒要唤醒一次”，新供货的STC15L2K60S2系列D版本芯片不存在该问题；
（6）新增1.3节“STC15W104系列单片机的介绍”；
（7）在附录K中新增STC15系列学习板4的实物图；
（8）STC15F2K60S2系列单片机与STC15F408AD系列单片机的工作电压调整为5.5V～4.2V；
（9）新增附录P“STC公司简介及招聘”；

2014-5-10更新内容：

（1）更新了彩色宣传资料；
（2）新增11.8节“用CCP/PCA高速脉冲输出功能实现3路9~16位PWM的测试程序”；；
（3）新增11.11节“利用T0的时钟输出功能实现8~16位PWM的测试程序”；
（4）新增11.12节“利用T1的时钟输出功能实现8~16位PWM的测试程序”；
（5）新增11.13节“利用T2的时钟输出功能实现8~16位PWM的测试程序”；
（6）新增8.17节“利用T2结合INT4模拟一个半双工串口的测试程序”；
（7）新增8.18节“利用两路CCP/PCA模拟一个全双工串口的测试程序”；

STC MCU Limited

2014-4-12更新内容：

（1）更新了彩色宣传资料；
（2）将第16章“使用STC15系列单片机的ADC做电容感应触摸按键”调整为第13章；
（3）STC15F/L104W系列、STC15F/L2K60S2系列、STC15F/L408AD系列及STC15W408AS系列单片机可用MOVC指令读取程序区和EEPROM区的数据；而STC15W204S系列、STC15W1K16S系列及STC15W408S系列单片机只能用MOVC读取程序区,不能读取EEPROM区的数据,否则程序会复位。

2014-3-31更新内容：

（1）更新了彩色宣传资料；
（2）更新了STC15F2K60S2系列单片机的价格；
（3）新增第16章“使用STC15系列单片机的ADC做电容感应触摸按键”；
（4）新增13.8节“利用STC15系列单片机SPI的主模式读写外部串行Flash”；
（5）新增10.9节"利用BandGap电压精确测量外部输入电压值及测试程序"；
（6）新增12.5节“在比较器负端产生不同的电压由比较器正端进行比较”；
（7）STC15W201S系列新增8-pin的封装为SOP8的单片机

2014-3-21更新内容：

（1）更新了彩色宣传资料；
（2）STC15W408AS系列所有型号单片机新增TSSOP28封装，封装为TSSOP28的单片机比封装为SOP28的单片机贵0.1元；
（3）IAP15W1K29S型号单片机新增TSSOP20封装，该型号的封装为TSSOP20的单片机与其封装为SOP28的单片机的价格相同；
（4）IAP15F2K61S2型号单片机新增TSSOP20封装，该型号的封装为TSSOP20的单片机与其封装为SOP28的单片机的价格相同；
（5）新增TSSOP28封装尺寸图；
（6）新增TSSOP20封装尺寸图；
（7）新增第15章“利用其他MCU对STC15系列单片机进行串口ISP下载”

2014-3-11更新内容：

（1）更新了彩色宣传资料
（2）STC15W408AS系列已开始供货，STC15W1K16S系列大批量供货中，STC15W408S系列供货中，STC15W4K60S4系列2014年6月开始送样。
（3）将文中输写错误“TOCLKO”更正为“T0CLKO”
（4）STC15W408S系列及STC15W408AS系列中以STC开头的单片机可以用MOVC读EEPROM.
（5）暂无STC15W2K60S2系列，删除原文中STC15W2K60S2系列的介绍

STC MCU Limited

2014-2-23更新内容：

（1）更新了彩色宣传资料
（2）新增了14.2.12节“用户接口”介绍
（3）将“程序加密后传输，收到用户命令后ISP下载”功能修正为“程序加密后传输”功能
（4）STC15W408S及STC15W1K16S系列单片机已开始供货，且要用其T0CLKO时钟输出功能需将P3.5设置为强推挽输出模式
（5）STC15W201S、STC15W408S及STC15W1K16S系列中以STC开头的单片机不能用MOVC读EEPROM.

2013-12-16更新内容：

（1）新增了章节14.5“如何让传统8051单片机学习板可仿真”
（2）更新了STC15W4K60S4系列的管脚图
（3）推荐了三本单片机教材
（4）新增STC15W408AS系列彩色宣传资料

2013-11-21更新内容：

（1）新增DFN-8封装尺寸图
（2）STC15F/L104W及STC15F/L105W型号单片机新提供DFN-8封装。

2013-10-18更新内容：

（1）STC15W408S系列和STC15W1K16S系列单片机无管脚XTAL2及管脚XTAL1
（2）STC15F2K60S2系列中RxD2管脚也不能唤醒掉电模式/停机模式
（3）STC15F408AD系列中RxD管脚不能唤醒掉电模式/停机模式
（4）STC15系列中串口1的切换寄存器位[S1_S1, S1_S0]的默认值由[0, 1]修改为[0, 0]，即串口1默认不在[P3.6/RxD_2, P3.7/TxD_2]上，若用户需要将串口1放在[P3.6/RxD_2, P3.7/TxD_2]上，则须在程序中将[S1_S1, S1_S0]的值设置为[0, 1]，且还须在用STC-ISP下载编程软件烧录用户程序时勾选中该软件的【硬件选项】工具栏中的选项【串口1数据线[RxD, TxD]从[P3.0, P3.1]切换到[P3.6, P3.7]，P3.7脚输出P3.6脚的输入电平】
（5）新增小节1.15“现供货的STC15系列为实现的计划功能”；
（6）新增“简易脱机下载器U7的使用说明”；
（7）修正12章“比较器内部规划图”中的错误；
（8）原研发顾问QQ13922805190容量已满，现将研发顾问QQ更为：800003751

STC MCU Limited

附录R：STC彩色宣传资料

R.1 STC15F2K60S2系列彩色宣传资料
R.2 STC15W1K16S系列彩色宣传资料
R.3 STC15W408AS系列彩色宣传资料
R.4 STC15W104系列彩色宣传资料
R.6 STC12C5A60S2系列彩色宣传资料
R.7 STC11/10系列带外部数据总线的彩色宣传资料
R.8 STC11/10系列无外部数据总线的彩色宣传资料
R.9 STC12C5201AD系列彩色宣传资料
R.10 STC12C5620AD系列彩色宣传资料
R.11 STC12C5410AD/STC12C2052AD系列彩色宣传资料
R.12 STC89C51/STC90C51系列彩色宣传资料
R.13 STC15W4K60S4系列彩色宣传资料

STC MCU Limited

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC15F2K60S2系列1T 8051单片机，2K字节SRAM，超高速双串口，高速A/D

不需外部晶振的单片机
不需外部复位的单片机

送仿真器

全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

采用宏晶第八代加密技术，现悬赏10万元人民币请专家帮忙查找加密有无漏洞

STC15F101W系列，大批量供货中，RMB0.99元起(STC15F100W)

每片单片机具有全球唯一
身份证号码(ID号)，
无法解密，加密坚不可摧
提供客制化IC设计服务

封装形式：
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产但不推荐)
LQFP32(9mm x 9mm, 推荐)
SOP28(强烈推荐)
SKDIP28(正常生产)
TSSOP20(6.5mm x 6.5mm)

全部175℃
8小时高温烘烤

型号	工作电压(V)	Flash程序存储器(byte)	大容量SRAM字节	串行口	SPI仅有主机模式	普通定时器计数器T0-T2外部管脚也能掉电唤醒	CCP PCA PWM可当外部中断并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	TSSOP20	SOP28	SKDIP28	LQFP32	PDIP40	LQFP44
																				所有封装 LQFP44/PDIP40 LQFP32 SOP28/SKDIP28/TSSOP20 (现此系列未生产PLCC44和SOP32封装) 部分封装价格(RMB ¥)					
大批量现货供应中	STC15F2K60S2系列单片机选型价格一览表，另有STC15L系列(工作电压2.4V-3.6V) 特别提醒：3路CCP/PCA/PWM还可当3路定时器使用																								
STC15F2K08S2	5.5-4.2	8K	2K	2	有	3	3-ch	有	5	10位	2	53K	有	有	8级	有	是	有	是		¥3.8	¥4.0	¥3.9	¥4.5	¥4.0
STC15F2K16S2	5.5-4.2	16K	2K	2	有	3	3-ch	有	5	10位	2	45K	有	有	8级	有	是	有	是		¥4.0	¥4.2	¥4.1	¥4.7	¥4.2
STC15F2K32S2	5.5-4.2	32K	2K	2	有	3	3-ch	有	5	10位	2	29K	有	有	8级	有	是	有	是		¥4.3	¥4.5	¥4.4	¥4.9	¥4.5
STC15F2K40S2	5.5-4.2	40K	2K	2	有	3	3-ch	有	5	10位	2	21K	有	有	8级	有	是	有	是		¥4.4	¥4.6	¥4.5	¥4.9	¥4.6
STC15F2K48S2	5.5-4.2	48K	2K	2	有	3	3-ch	有	5	10位	2	13K	有	有	8级	有	是	有	是		¥4.4	¥4.6	¥4.5	¥4.9	¥4.6
STC15F2K56S2	5.5-4.2	56K	2K	2	有	3	3-ch	有	5	10位	2	5K	有	有	8级	有	是	有	是		¥4.4	¥4.6	¥4.5	¥4.9	¥4.6
STC15F2K60S2	5.5-4.2	60K	2K	2	有	3	3-ch	有	5	10位	2	1K	有	有	8级	有	是	有	是		¥4.4	¥4.6	¥4.5	¥4.9	¥4.6
IAP15F2K61S2 本身就是仿真器	5.5-4.2	61K	2K	2	有	3	3-ch	有	5	10位	2	IAP	有	有	8级	有	是	有	是	¥4.4	¥4.4	¥4.6	¥4.5	¥4.9	¥4.6
																				用户可将用户程序区的程序Flash当EEPROM使用					
IRC15F2K63S2 使用外部晶振或内部24MHz时钟	5.5-4.2	63.5K	2K	2	有	3	3-ch	有	5	10位	2	IAP	有	有	固定	有	是	无	否					¥4.9	¥4.6
																				用户可将用户程序区的程序Flash当EEPROM使用					
IAP15F2K61S	5.5-4.2	61K	2K	1	有	3	-	有	5	–	2	IAP	有	有	8级	有	是	有	是					¥4.5	¥4.0
																				用户可将用户程序区的程序Flash当EEPROM使用					
大批量供货中	STC15F101W系列单片机选型价格一览表，另有STC15L系列(工作电压2.4V-3.6V) 串行口功能可由[P3.0/INT4, P3.1]结合定时器实现																			SOP8	DIP8	DFN8			
STC15F100W	5.5-3.8	0.5K	128	-	-	2(T0/T2)	–	有	5	–	1	–	有	有	8级	有	是	有	是	¥0.99					
STC15F101W	5.5-3.8	1K	128	-	-	2(T0/T2)	–	有	5	–	1	4K	有	有	8级	有	是	有	是	¥1.2					
STC15F102W	5.5-3.8	2K	128	-	-	2(T0/T2)	–	有	5	–	1	3K	有	有	8级	有	是	有	是	¥1.3	¥1.5				
STC15F104W	5.5-3.8	4K	128	-	-	2(T0/T2)	–	有	5	–	1	1K	有	有	8级	有	是	有	是	¥1.4	¥1.6	¥1.5			
IRC15F107W 固定使用内部24MHz时钟	5.5-3.8	7K	128	-	-	2(T0/T2)	–	有	5	–	1	IAP	有	有	固定	有	是	无	否	¥1.4	¥1.6				
																				用户可将用户程序区的程序Flash当EEPROM使用					

以上单价为10K/M起定量，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

特别声明：以15F和15L开头且有SPI功能的芯片不支持"SPI从机模式"；以15W开头且有SPI功能的芯片，SPI主/从机模式均支持。

IRC15F2K63S2和IRC15L2K63S2型号单片机的内部复位门槛电压固定，P5.4不可当复位管脚RST使用，[XTAL2/P1.6, XTAL1/P1.7]不可当I/O口使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能。

STC 单片机在线编程/ISP及仿真线路(USB转串口)

建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]
大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

宏晶•STC15F2K60S2系列主要性能：

- 大容量2048字节片内RAM数据存储器
- 高速：1个时钟/机器周期，增强型8051内核，速度比传统8051快7～12倍
  速度也比STC早期的1T系列单片机(如STC12/11/10系列)的速度快20%
- 宽电压：5.5～4.2V，2.4～3.6V(STC15L2K60S2系列)
- 低功耗设计：低速模式，空闲模式，掉电模式(可由外部中断或内部掉电唤醒定时器唤醒)
- 不需外部复位的单片机，ISP编程时8级复位门槛电压可选，内置高可靠复位电路
- 不需外部晶振的单片机，ISP编程时内部时钟从5MHz~35MHz可设(相当于普通8051：60～420MHz)
  内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
- 支持掉电唤醒的资源有：INT0/INT1(上升沿/下降沿中断均可)，INT2/INT3/INT4(下降沿中断)；CCP0/CCP1/CCP2/T0/T1/T2管脚；内部掉电唤醒专用定时器
- 8/16/24/32/40/48/56/60/61/63.5K字节片内Flash程序存储器，擦写次数10万次以上
- 大容量片内EEPROM功能，擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- 高速ADC，8通道10位，速度可达30万次/秒，3路PWM还可当3路D/A使用
- 3通道捕获/比较单元(CCP/PCA/PWM)
  ----也可用来再实现3路D/A或3个定时器或3个外部中断(支持上升沿/下降沿中断)
- 利用CCP/PCA高速脉冲输出功能可实现3路9～16位PWM (每通道占用系统时间小于0.6%)
- 利用定时器T0、T1或T2的时钟输出功能可实现高精度的8～16位PWM (占用系统时间小于0.4%)
- 6个定时器，2个16位可重装载定时器T0和T1兼容普通8051的定时器，新增了一个16位的定时器T2，并都可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出(÷1或÷2或÷4)，3路CCP/PCA可再实现3个定时器
- 可编程时钟输出功能(对内部系统时钟或外部管脚的时钟输入进行时钟分频输出)：
  ① T0在P3.5输出时钟；② T1在P3.4输出时钟；③ T2在P3.0输出时钟，以上3个定时器/计数器输出时钟均可1~65536级分频输出；④ 内部主时钟在P5.4/MCLKO对外输出时钟(STC15系列8-pin单片机的主时钟在P3.4/MCLKO对外输出时钟)
- 超高速双串口/UART，两个完全独立的高速异步串行通信端口，分时切换可当5组串口使用
- SPI高速同步串行通信接口
- 硬件看门狗(WDT)
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(42/38/30/26/18个)，复位后为：准双向口/弱上拉(8051传统I/O口)
  可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
  每个I/O口驱动能力均可达到20mA，但整个芯片最大不要超过120mA
- 如果I/O口不够用可以用3根普通I/O口线外接74HC595(¥0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D作按键扫描来节省I/O口

选择宏晶•STC15F2K60S2系列单片机理由：

★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
★ 片内大容量2048字节SRAM
★ 无法解密，宏晶第八代加密技术，现悬赏10万元人民币请专家帮忙查找加密有无漏洞
★ 超强抗干扰：
1. 高抗静电(ESD保护)整机轻松过2万伏静电测试　2. 轻松过4kV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动　4. 宽温度范围，-40℃~+85℃
★ 大幅降低EMI，内部可配置时钟，1个时钟/机器周期，
可用低频时钟----出口欧美的有力保证
★ 超低功耗：
1. 掉电模式：外部中断唤醒功耗 <0.1uA
2. 空闲模式：典型功耗<1mA，　3. 正常工作模式：4mA～6mA
4. 掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用于电池供电系统，如水表、气表等
★ 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元)，1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

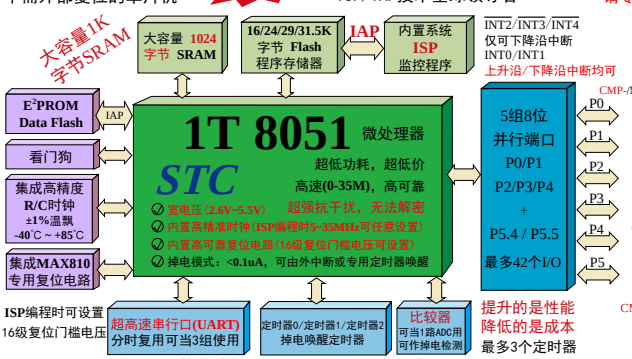
官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC15W1K16S 系列 1T 8051 单片机，宽电压，超高速串行口，比较器

不需外部晶振的单片机
不需外部复位的单片机

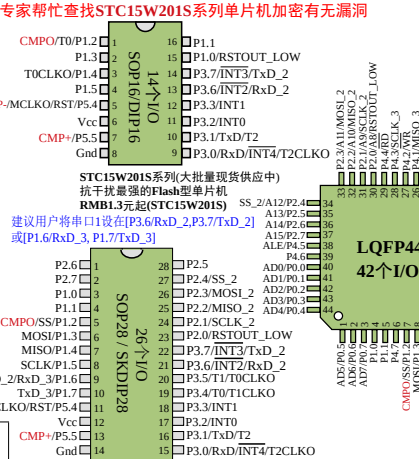
送仿真器

全球第一款真正意义上的单片机ISP/IAP技术全球领导者



提升的是性能
降低的是成本
最多3个定时器

采用宏晶第九代加密技术，现悬赏20万元人民币
请专家帮忙查找STC15W201S系列单片机加密有无漏洞



封装形式：
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产但不推荐)
LQFP32(9mm x 9mm, 推荐)
SOP28(强烈推荐)
SKDIP28(正常生产)
TSSOP20(6.5mm x 6.5mm)
SOP16(6mm x 9mm, 强烈推荐)
DIP16(正常生产)

全部175℃ 8小时高温烘烤

STC15W201S系列是目前全球抗干扰最强的Flash型单片机
现悬赏20万元人民币寻找比STC15W201S系列抗干扰更强的Flash型单片机

型号	工作电压(V)	Flash程序存储器(byte)	大容量SRAM字节	串行口	并行口可当SPI	掉电唤醒	普通定时器	CCP PCA	掉电唤醒	标准外部中断	支持掉电唤醒	A/D (3路PWM可当3路D/A使用)	比较器(可当1路ADC使用，可作外部掉电检测)	EEPROM	内部低压检测中断	内部掉电唤醒	内部高可靠复位(可选复位门槛电压)	可外加掉电复位电路	程序可外加密钥	支持RS485下载	封装
STC15W201S	2.4-5.5	1K	256	1			T0 / T2	–	有	5	–	有	有	4K	有	有	16级	有	是	有	是
STC15W202S	2.4-5.5	2K	256	1			T0 / T2	–	有	5	–	有	有	3K	有	有	16级	有	是	有	是
STC15W203S	2.4-5.5	3K	256	1			T0 / T2	–	有	5	–	有	有	2K	有	有	16级	有	是	有	是
STC15W204S	2.4-5.5	4K	256	1			T0 / T2	–	有	5	–	有	有	1K	有	有	16级	有	是	有	是

所有封装 LQFP44/PDIP40 LQFP32 SOP28/SKDIP28 TSSOP20 SOP16/DIP16

封装价格(RMB ¥)	SOP8	SOP16	DIP16	-	-
STC15W201S系列单片机选型价格一览表，大批量现货供应中	¥1.6	¥1.7	¥1.9	-	-
IAP15W205S 2.4-5.5 5K 256 1 T0 / T2 – 有 5 – 有 IAP 有 有 16级 有 是 有 是	¥1.6	¥1.7	¥1.9	-	-
用户可将用户程序区的程序Flash当EEPROM使用					
IRC15W207S 固定使用内部24MHz时钟 2.4-5.5 7.5K 256 1 T0 / T2 – 有 5 – 有 IAP 有 有 固定 有 是 无 否	¥1.6	¥1.7	¥1.9	-	-
用户可将用户程序区的程序Flash当EEPROM使用					

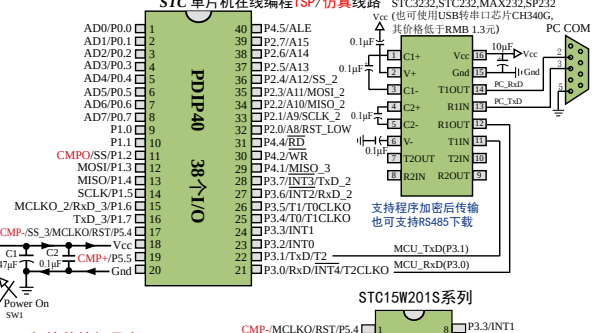
STC15W408S系列单片机选型价格一览表，大批量现货供应中	SOP28	SKDIP28	LQFP32	PDIP40	LQFP44
STC15W404S 2.4-5.5 4K 512 1 有 3 – 有 5 – 有 9K 有 有 16级 有 是 有 是	¥2.3	¥2.5	¥2.4	¥3.0	¥2.5
STC15W408S 2.4-5.5 8K 512 1 有 3 – 有 5 – 有 5K 有 有 16级 有 是 有 是	¥2.5	¥2.7	¥2.6	¥3.0	¥2.7
STC15W410S 2.4-5.5 10K 512 1 有 3 – 有 5 – 有 3K 有 有 16级 有 是 有 是					¥2.8
IAP15W413S 2.4-5.5 13K 512 1 有 3 – 有 5 – 有 IAP 有 有 16级 有 是 有 是	¥2.6	¥2.8	¥2.7	¥3.3	¥2.8
用户可将用户程序区的程序Flash当EEPROM使用	¥2.6	¥2.8	¥2.7	¥3.3	¥2.8
IRC15W415S 固定使用内部24MHz时钟 2.4-5.5 15.5K 512 1 有 3 – 有 5 – 有 IAP 有 有 固定 有 是 无 否					
用户可将用户程序区的程序Flash当EEPROM使用					

STC15W1K16S系列单片机选型价格一览表，大批量现货供应中	TSSOP20	SOP28	SKDIP28	LQFP32	PDIP40	LQFP44
STC15W1K16S 2.6-5.5 16K 1K 1 有 3 – 有 5 – 有 13K 有 有 16级 有 是 有 是		¥3.3	¥3.5	¥3.4	¥4.0	¥3.5
STC15W1K24S 2.6-5.5 24K 1K 1 有 3 – 有 5 – 有 5K 有 有 16级 有 是 有 是	¥3.6	¥3.6	¥3.8	¥3.7	¥4.3	¥3.8
IAP15W1K29S 2.6-5.5 29K 1K 1 有 3 – 有 5 – 有 IAP 有 有 16级 有 是 有 是	¥3.6	¥3.6	¥3.8	¥3.7	¥4.3	¥3.8
用户可将用户程序区的程序Flash当EEPROM使用						
IRC15W1K31S 固定使用内部24MHz时钟 2.6-5.5 31.5K 1K 1 有 3 – 有 5 – 有 IAP 有 有 固定 有 是 无 否	¥3.6	¥3.8	¥3.7	¥4.3	¥3.8	
用户可将用户程序区的程序Flash当EEPROM使用						

以上单价为10K/M起定量，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

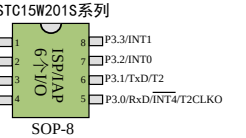
程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

上表中以IRC开头的单片机固定使用内部24MHz时钟，其内部复位门槛电压固定，P5.4不可当复位管脚RST使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能。



每片单片机具有
全球唯一身份证号码(ID号)，
无法解密，加密坚不可摧

提供客制化IC设计服务



- 硬件看门狗(WDT)
- 超高速异步串行通信端口/UART, 分时切换可当3组串口使用SPI高速同步串行通信接口
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口 (42/38/30/26/18个)，**复位后为：准双向口/弱上拉** (8051传统I/O口)
  可设置四种模式：准双向口/弱上拉，**强推挽/强上拉，仅为输入/高阻，开漏**
  每个I/O口驱动能力均可达到20mA，但整个芯片最大不要超过120mA
- 如果I/O口不够用可以用3根普通I/O口线外接74HC595(¥0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D作按键扫描来节省I/O口

大陆本土宏晶STC姚永平独立创新设计，请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

宏晶•STC15W1K16S系列主要性能：

- 大容量1024字节片内RAM数据存储器
- 高速：1个时钟/机器周期，增强型8051内核，速度比**传统**8051快7～12倍
  速度**也比**STC早期的1T系列单片机(如STC12/11/10系列)的速度快20%
- 宽电压：2.6V ～ 5.5V
- 低功耗设计：低速模式，**空闲**模式，**掉电模式**(可由外部中断**或内部掉电唤醒定时器唤醒**)
- 不需外部复位的单片机，ISP编程时16级复位门槛电压可选，内置高可靠复位电路
- 不需外部晶振的单片机，ISP编程时内部时钟从5MHz~35MHz可设(**相当于普通**8051：60～420MHz)
  内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
- 支持掉电唤醒的资源有：INT0/INT1(上升沿/下降沿中断均可), INT2 / INT3 / INT4(下降沿中断)；
  RxD/T0/T1/T2管脚：内部掉电唤醒专用定时器
- 16/24/29K/31.5K字节片内Flash程序存储器，擦写次数10万次以上
- 大容量片内EEPROM功能, 擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程, 无需编程器/仿真器
- 比较器，可当1路ADC使用，并可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出（可设置极性），也支持外部管脚CMP+与内部参考电压进行比较
- 3个16位可重装载定时器/计数器(T0/T1/T2，其中T0/T1兼容普通8051的定时器/计数器)，并都可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4）

选择宏晶•STC15W1K16S系列单片机理由：

★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
★ 片内大容量1024字节SRAM
★ 无法解密，宏晶第九代加密技术，
现悬赏20万元人民币请专家帮忙查找STC15W201S系列和STC15W1K16S系列加密有无漏洞
★ 超强抗干扰：
1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
2. 轻松过4kV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动
4. 宽温度范围，-40℃~+85℃
★ 大幅降低EMI，内部可配置时钟，1个时钟/机器周期，
可用低频时钟一一出口欧美的有力保证
★ 超低功耗：
1. **掉电模式：外部中断唤醒功耗** <0.1uA
2. 空闲模式：典型功耗<1mA，
3. 正常工作模式：4mA ~ 6mA
4. **掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用于电池供电系统，如水表、气表、便携设备等**
★ 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元)，1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

宏晶•STC15W408AS 系列 1T 8051 单片机，超高速串口，高速A/D，比较器，宽电压

不需外部晶振的单片机
不需外部复位的单片机

送仿真器

全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

采用宏晶第九代加密技术，现悬赏**20**万元人民币请专家帮忙查找加密有无漏洞

提供客制化
IC设计服务

INT2/INT3/INT4
仅可下降沿中断
INT0/INT1
上升沿/下降沿中断均可

ISP编程时可设置
16级复位门槛电压
最多5个定时器

提升的是性能
降低的是成本
比较器可作掉电检测

封装形式：
SOP28(强烈推荐)
TSSOP28(6.4mm x 9.7mm)
SKDIP28(正常生产)
QFN28(5mm x 5mm)
SOP20(强烈推荐)
DIP20(正常生产)
TSSOP20 (6.5mm x 6.5mm)
SOP16(6mm x 9mm, 强烈推荐)
DIP16(正常生产)

全部175℃
8小时高温烘烤

建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]
或[P1.6/RxD_3, P1.7/TxD_3]

STC 单片机在线编程/**ISP及仿真**线路(*USB*转串口)

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路(3路PWM可当3路D/A使用)	比较器(可当1路ADC使用,可作掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部高可靠复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	SOP16	DIP16	SOP20	DIP20	TSSOP20	SOP28	SKDIP28	TSSOP28	QFN28
																					所有封装 SOP28/ SKDIP28 / TSSOP28 / QFN28 SOP20 / DIP20 / TSSOP20 SOP16 / DIP16；部分封装 价格(RMB ¥)								
STC15W408AS系列单片机选型价格一览表，大批量现货供应中；特别提示：3路CCP/PCA/PWM还可当3路定时器使用																													
STC15W401AS	2.4-5.5	1K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16级	有	是	有	是	¥1.60	¥1.80	¥1.80	¥2.00	¥1.90	¥2.00	¥2.20	¥2.10	¥2.20
STC15W402AS	2.4-5.5	2K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16级	有	是	有	是	¥1.80	¥2.00	¥2.00	¥2.20	¥2.10	¥2.20	¥2.40	¥2.30	¥2.40
STC15W404AS	2.4-5.5	4K	512	1	有	2	3-ch	有	5	10-bit	有	1	9K	有	有	16级	有	是	有	是	¥1.99	¥2.20	¥2.20	¥2.40	¥2.30	¥2.50	¥2.70	¥2.60	¥2.70
STC15W408AS	2.4-5.5	8K	512	1	有	2	3-ch	有	5	10-bit	有	1	5K	有	有	16级	有	是	有	是	¥2.20	¥2.40	¥2.40	¥2.60	¥2.50	¥2.70	¥2.90	¥2.80	¥2.90
IAP15W413AS	2.4-5.5	13K	512	1	有	2	3-ch	有	5	10-bit	有	1	IAP	有	有	16级	有	是	有	是	¥2.30	¥2.50	¥2.50	¥2.70	¥2.60	¥2.80	¥3.00	¥2.90	¥3.00
																					用户可将用户程序区的程序Flash当EEPROM使用								
IRC15W415AS 使用外部晶振或内部24MHz时钟	2.4-5.5	15.5K	512	1	有	2	3-ch	有	5	10-bit	有	1	IAP	有	有	固定	有	是	无	否	¥2.30	¥2.50	¥2.50	¥2.70	¥2.60	¥2.80	¥3.00	¥2.90	¥3.00
																					用户可将用户程序区的程序Flash当EEPROM使用								

以上单价为10K/M起定量，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

IRC15W415AS型号单片机的内部复位门槛电压固定，P5.4不可当复位管脚RST使用，[XTAL2/P1.6, XTAL1/P1.7]不可当I/O口使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能 。

STC15F101W系列，大批量供货中
RMB0.99元起(STC15F100W)

SOP-8/DIP-8

大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无. . .

宏晶•STC15W408AS系列主要性能：

每片单片机具有
全球唯一身份证号码(ID号)
无法解密，加密坚不可摧

- 512字节片内RAM数据存储器
- 高速：1个时钟/机器周期，增强型8051内核，速度比传统8051快7～12倍
  速度也比STC早期的1T系列单片机(如STC12/11/10系列)的速度快20%
- 宽电压：2.4V ～ 5.5V
- 低功耗设计：低速模式，空闲模式，掉电模式(可由外部中断或内部掉电唤醒定时器唤醒)
- 不需外部复位的单片机，ISP编程时16级复位门槛电压可选，内置高可靠复位电路
- 不需外部晶振的单片机，ISP编程时内部时钟从5MHz~35MHz可设(相当于普通8051：60～420MHz)
  内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
- 支持掉电唤醒的资源有：INT0/INT1(上升沿/下降沿中断均可), INT2/INT3/INT4(下降沿中断)；
  CCP0/CCP1/CCP2/RxD/T0/T2管脚；内部掉电唤醒专用定时器
- 4K/8K/13K/15.5K字节片内Flash程序存储器，擦写次数10万次以上
- 大容量片内EEPROM功能, 擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程, 无需编程器/仿真器
- 高速ADC，8通道10位，速度可达30万次/秒，3路PWM还可当3路D/A使用
- 比较器，可当1路ADC使用，并可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出(可设置极性)，也支持外部管脚CMP+与内部参考电压进行比较
- 3通道捕获/比较单元(CCP/PCA/PWM)
  ----也可用来再实现3路D/A或3个定时器或3个外部中断(支持上升沿/下降沿中断)
- 利用CCP/PCA高速脉冲输出功能可实现3路9 ~ 16位PWM (每通道占用系统时间小于0.6%)
- 利用定时器T0的时钟输出功能可实现高精度的8 ~ 16位PWM (占用系统时间小于0.4%)
- 5个定时器，2个16位可重装载定时器/计数器(T0/T2，其中T0兼容普通8051的定时器/计数器)，并都可实现可编程时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出(÷1或÷2或÷4)，3路CCP/PCA可再实现3个定时器
- 可编程时钟输出功能(对内部系统时钟或外部管脚的时钟输入进行时钟分频输出)：
  ① T0在P3.5输出时钟；② T2在P3.0输出时钟，以上2个定时器/计数器输出时钟均可1~65536级分频输出；③ 内部主时钟在P5.4/MCLKO对外输出时钟(STC15系列8-pin单片机的主时钟在P3.4/MCLKO对外输出时钟)
- 硬件看门狗(WDT)
- 超高速异步串行通信端口/UART, 分时切换可当3组串口使用
- SPI高速同步串行通信接口
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(26/18/14个)，复位后为：准双向口/弱上拉(8051传统I/O口)
  可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
  每个I/O口驱动能力均可达到20mA，但整个芯片最大不要超过120mA

选择宏晶•STC15W408AS系列单片机理由：

★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
★ 无法解密，宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞
★ 超强抗干扰：
　1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
　2. 轻松过4kV快速脉冲干扰(EFT测试)
　3. 宽电压，不怕电源抖动
　4. 宽温度范围，-40℃~+85℃
★ 大幅降低EMI，内部可配置时钟，1个时钟/机器周期，
　可用低频时钟----出口欧美的有力保证
★ 超低功耗：
　1. **掉电模式:外部中断唤醒功耗** <0.1uA
　2. 空闲模式：典型功耗<1mA,　　3. 正常工作模式:4mA ~ 6mA
　4. 掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用于电池供电系统，如水表、气表、便携设备等
★ 利用CCP/PCA高速脉冲输出功能可实现3路9 ~ 16位PWM (每通道占用系统时间小于0.6%)
★ 利用定时器T0的时钟输出功能可实现高精度的8 ~ 16位PWM (占用系统时间小于0.4%)
★ 在系统可仿真，在系统可编程，**无需专用编程器, 无需专用仿真器，可远程升级**
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元), 1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，无法解密

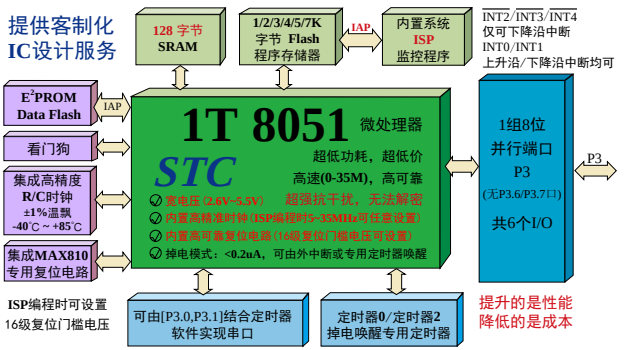
8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

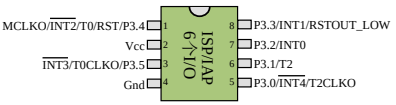
宏晶•STC15W104系列新一代 1T 8051 单片机，宽电压，高可靠，超低价

不需外部晶振的单片机
不需外部复位的单片机
送仿真器
全球第一款真正意义上的单片机ISP/IAP技术全球领导者
采用宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞

提供客制化IC设计服务



STC15W104系列，已开始大批量供货，RMB1.25元起(STC15W101)



SOP-8/DIP-8/DFN-8

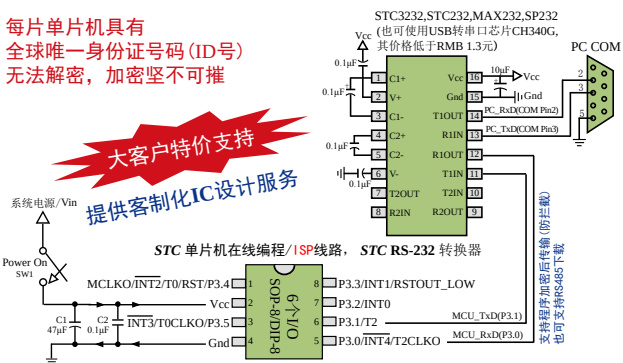
封装形式：
SOP-8/DIP-8/DFN-8
强烈推荐选择SOP-8贴片封装

全部175℃ 8小时高温烘烤

注意：STC15W104系列与STC15F104E系列是不同的两个系列，STC15W104系列采用STC-Y5超高速CPU内核，在相同的时钟频率下，速度比STC早期的1T系列单片机(如STC12/STC11/STC10系列)的速度快20%，而STC104E系列采用的是STC-Y3 CPU内核，速度没有STC15WF104系列的速度快。

每片单片机具有
全球唯一身份证号码(ID号)
无法解密，加密坚不可摧

大客户特价支持
提供客制化IC设计服务



型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口	SPI	普通定时器计数器T0/T2外部管脚也能掉电唤醒	CCP PCA PWM D/A	掉电唤醒定时器	标准外部中断支持掉电唤醒	A/D 8路	看门狗	内部低压检测中断	内部复位(可选复位门槛电压)	内部高精准时钟	可对外输出时钟及复位	程序加密后传输(防拦截)	支持RS485下载	封装8-Pin SOP-8	DIP-8 / DFN-8 (6个I/O口) 价格(RMB ¥) DIP-8	DFN-8
STC15W104系列单片机选型价格一览表，此系列大批量现货供应中 串行口功能可由[P3.0/INT4, P3.1]结合定时器实现																					
STC15W100	5.5-2.4	512	128	无	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.05	–	
STC15W101	5.5-2.4	1K	128	4K	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.25	–	
STC15W102	5.5-2.4	2K	128	3K	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.35	¥1.55	
STC15W104	5.5-2.4	4K	128	1K	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.45	¥1.65	¥1.55
IAP15W105	5.5-2.4	5K	128	IAP	无	无	2	无	有	5个	无	有	有	16级	有	是	有	是	¥1.45	¥1.65	¥1.55
IRC15W107 固定使用内部24MHz时钟	5.5-2.4	7K	128	IAP	无	无	2	无	有	5个	无	有	有	固定	有	是	无	否	¥1.45 用户可将用户程序区的程序FLASH当EEPROM使用	¥1.65 用户可将用户程序区的程序FLASH当EEPROM使用	
STC15F101W系列单片机选型价格一览表，该系列低压型号(工作电压2.4V-3.6V)建议用STC15W104系列代替 串行口功能可由[P3.0/INT4, P3.1]结合定时器实现 此系列大批量现货供应中																					
STC15F100W	5.5-3.8	512	128	无	无	无	2	无	有	5个	无	有	有	8级	有	是	有	是	¥0.99	–	
STC15F101W	5.5-3.8	1K	128	4K	无	无	2	无	有	5个	无	有	有	8级	有	是	有	是	¥1.2		
STC15F104W	5.5-3.8	4K	128	1K	无	无	2	无	有	5个	无	有	有	8级	有	是	有	是	¥1.4	¥1.6	¥1.5
IRC15F107W 固定使用内部24MHz时钟	5.5-3.8	7K	128	IAP	无	无	2	无	有	5个	无	有	有	固定	有	是	无	否	¥1.4 用户可将用户程序区的程序FLASH当EEPROM使用	¥1.6	

以上单价为10K/M起定量，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

IRC15W107型号单片机固定使用内部24Mhz时钟，其内部复位门槛电压固定，P5.4不可当复位管脚RST使用，P3.2/P3.3与下载无关，且不支持"程序加密后传输"功能。

宏晶•STC15W104系列主要性能：

- 高速：1个时钟/机器周期，增强型8051内核，速度比普通8051快6～12倍
  速度也比STC早期的1T系列单片机(如STC12/11/10系列)的速度快20%
- 宽电压：5.5～2.4V
- 低功耗设计：低速模式，空闲模式，掉电模式(可由外部中断或内部掉电唤醒定时器唤醒)
- 不需外部复位的单片机，ISP编程时16级复位门槛电压可选，内置高可靠复位电路
- 不需外部晶振的单片机，ISP编程时内部时钟从5MHz~35MHz可设(相当于8051：60～420MHz)
  内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
- 支持掉电唤醒的资源有：INT0/INT1(上升沿/下降沿中断均可), INT2/INT3/INT4(下降沿中断)；T0/T2管脚：内部掉电唤醒专用定时器
- 1K/2K/3K/4K/5K/7K字节片内Flash程序存储器，擦写次数10万次以上
- 128字节片内RAM数据存储器
- 片内EEPROM功能, 擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程, 无需编程器/仿真器
- 2个16位可重装载定时器T0/T2，并可实现时钟输出功能，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4）
- 可编程时钟输出功能(对内部系统时钟或外部管脚的时钟输入进行时钟分频输出)：
  ① T0在P3.5输出时钟；② T2在P3.0输出时钟；③ 内部主时钟在P3.4/MCLKO对外输出时钟(STC15系列8-pin以上单片机的主时钟在P5.4/MCLKO对外输出时钟)

大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

- 硬件看门狗(WDT)
- 串口功能可由[P3.0/INT4, P3.1]结合定时器实现
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(8个)，复位后为：准双向口/弱上拉(8051传统I/O口)
  可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
  每个I/O口驱动能力均可达到20mA，但整个芯片最大不要超过90mA
- 如果I/O口不够用可以用3根普通I/O口线外接74HC595(¥0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D作按键扫描来节省I/O口

选择宏晶•STC15W104系列单片机理由：

★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
★ 不需外部晶振的单片机，内部集成高精度R/C时钟(±0.3%)，
  ±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
★ 不需外部复位的单片机，内部集成高可靠复位电路，
  ISP编程时16级复位门槛电压可选，当然也可以继续用外部复位电路
★ 无法解密，宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞
★ 超强抗干扰：
  1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
  2. 轻松过4kV快速脉冲干扰(EFT测试)
  3. 宽电压，不怕电源抖动　　4. 宽温度范围，-40℃~+85℃
★ 大幅降低EMI，内部可配置时钟，1个时钟/机器周期，可用低频时钟
  ----出口欧美的有力保证
★ 超低功耗：
  1. 掉电模式：外部中断唤醒功耗 <0.2uA
  2. 空闲模式：典型功耗 <1mA，　　3. 正常工作模式：4mA -6mA
  4. 掉电模式可由外部中断唤醒，适用于电池供电系统，如水表、气表、便携设备等
★ 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元)，1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

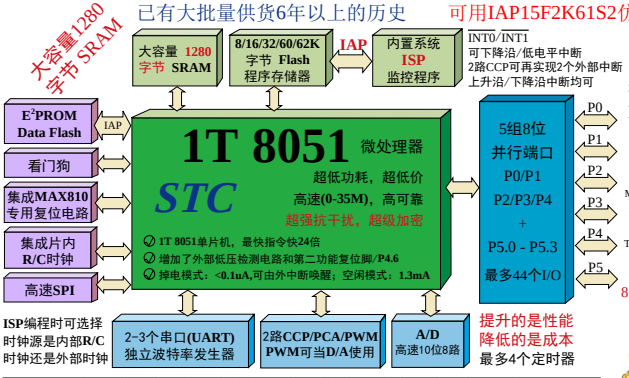
超强抗干扰，超级加密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC12C5A60S2系列 1T 8051 单片机，2-3个串口，2路CCP/PCA/PWM，高速A/D

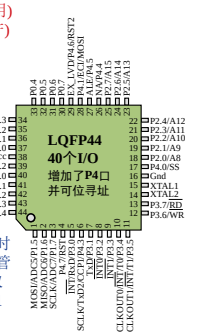
已有大批量供货6年以上的历史　可用IAP15F2K61S2仿真(仅供参考)



封装形式：
LQFP48(9x9mm)
LQFP44(12x12mm)
PDIP40
PLCC44(不推荐使用)
QFN-40(5x5, 未生产)

CCP：是英文单词的缩写
Capture(捕获)
Compare(比较)
PWM(脉宽调制)

提供客制化IC设计服务

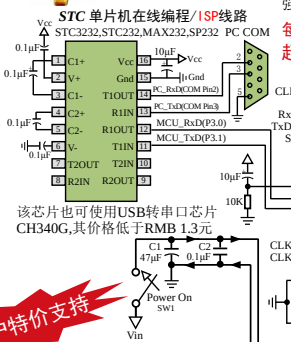


传统8051单片机划时代开级换代产品，管脚完全兼容，直接取代传统89C51/89S51系列单片机

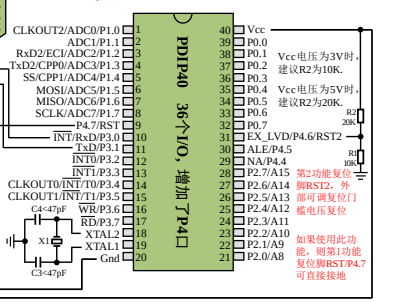
强烈推荐选择LQFP44, LQFP48贴片封装

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEP ROM	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T1外部管脚也能掉电唤醒	CCP PCA PWM可当外部中断并可掉电唤醒	独立波特率发生器	A/D 8路 25万次每秒(2路PWM还可当2路D/A使用)	看门狗	内置复位	外部实时低压检测中断	外部复位(可调整复位门槛电压)	LQFP48 LQFP44 PDIP40 不推荐PLCC44	LQFP48	LQFP44	PDIP40
															部分封装价格(RMB ¥)			
STC12C5A60S2系列单片机选型价格一览表 特别提醒：2路CCP/PCA/PWM还可当2路定时器使用，另有STC12LE系列(工作电压2.1V-3.6V)																		
STC12C5A08S2	5.5-4.0	8K	1280	53K	2-3	有	2	2-ch	有	2	10位	有	有	有	有		¥4.5	¥5.0
STC12C5A16S2	5.5-4.0	16K	1280	45K	2-3	有	2	2-ch	有	2	10位	有	有	有	有		¥4.7	¥5.2
STC12C5A32S2	5.5-4.0	32K	1280	29K	2-3	有	2	2-ch	有	2	10位	有	有	有	有		¥5.49	¥5.99
STC12C5A40S2	5.5-4.0	40K	1280	21K	2-3	有	2	2-ch	有	2	10位	有	有	有	有		¥5.99	¥6.49
STC12C5A48S2	5.5-4.0	48K	1280	13K	2-3	有	2	2-ch	有	2	10位	有	有	有	有		¥5.99	¥6.49
STC12C5A56S2	5.5-4.0	56K	1280	5K	2-3	有	2	2-ch	有	2	10位	有	有	有	有		¥5.99	¥6.49
STC12C5A60S2	5.5-4.0	60K	1280	1K	2-3	有	2	2-ch	有	2	10位	有	有	有	有		¥5.99	¥6.49
IAP12C5A62S2	5.5-4.0	62K	1280	IAP	2-3	有	2	2-ch	有	2	10位	有	有	有	有	用户可将用户程序区的程序Flash当EEPROM使用	¥5.99	¥6.49

以上单价为200K/M起定量，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价



每片单片机具有全球唯一身份证号码(ID号)
超级加密，加密坚不可摧



宏晶•STC12C5A60S2系列主要性能：

大客户特价支持

- 高速：1个时钟/机器周期，增强型8051内核，速度比普通8051快6～12倍
- 宽电压：5.5～4.0V，2.1～3.6V（STC12LE5A60S2系列）
- 增加第二复位功能脚/P4.6(高可靠复位，可调整复位门槛电压，频率<12MHz时，无需此功能)
- 增加外部掉电检测电路/P4.6，可在掉电时，及时将数据保存进EEPROM，正常工作时无需操作EEPROM
- 低功耗设计：空闲模式(可由任意一个中断唤醒)
- 低功耗设计：掉电模式(可由外部中断唤醒)，可支持下降沿/上升沿和远程唤醒
- 支持掉电唤醒的管脚：P3.2/INT0, P3.3/INT1, T0/P3.4, T1/P3.5, RxD/P3.0, P1.3/CCP0(或P4.2/CCP0), P1.4/CCP1（或P4.3/CCP1), EX_LVD/P4.6
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部R/C振荡器可选，在ISP下载编程用户程序时设置
- 8/16/32/40/48/56/60/62K字节片内Flash程序存储器，擦写次数10万次以上
- 1280字节片内RAM数据存储器
- 大容量片内EEPROM功能，擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- 8通道，10位高速ADC，速度可达25万次/秒，2路PWM还可当2路D/A使用
- 2通道捕获/比较单元(CCP/PCA/PWM)，
  ---也可用来再实现2个定时器或2个外部中断(支持上升沿/下降沿中断)
- 2个16位定时器(兼容普通8051定时器T0/T1)，2路PCA可再实现2个定时器
- 可编程时钟输出功能(T0在P3.4输出时钟，T1在P3.5输出时钟，BRT在P1.0输出时钟
- 硬件看门狗(WDT)
- 独立波特率发生器
- SPI高速同步串行通信接口
- 双串口，全双工异步串行口(UART)，兼容普通8051串口，分时复用可当3组使用
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(36/40/44个)，复位后为： 准双向口/弱上拉（普通8051传统I/O口）
  可设置成四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
  每个I/O口驱动能力均可达到20mA，但建议整个芯片不要超过120mA

大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

复位脚：烧录程序时如设置为I/O口，可当I/O口使用或浮空
EX_LVD：是外部低压检测中断/比较器
不用的I/O口：浮空即可

- 如果I/O口不够用可以用3根普通I/O口线外接74HC595(¥0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D作按键扫描来节省I/O口

选择宏晶•STC12C5A60S2系列单片机理由：

★ 超级加密，采用宏晶第六代加密技术
★ 超强抗干扰，超强抗静电，整机轻松过2万伏静电测试
★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
　----出口欧美的有力保证
★ 输入/输出口多，最多有44个I/O，A/D做按键扫描还可以节省很多I/O
★ 超低功耗：
　掉电模式：外部中断唤醒功耗 <0.1uA，支持下降沿/上升沿/低电平和远程唤醒
　适用于电池供电系统，如水表、气表、便携设备等。
　空闲模式：典型功耗<1.3mA,
　正常工作模式：2mA -7mA
★ 在系统可编程，无需编程器，无需仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元)，1万片/人/天，有自动烧录机接口
★ 内部集成高可靠复位电路，外部复位电路可彻底省掉，当然也可以继续用外部复位电路
★ 全部175℃，8小时高温烘烤，高可靠制造保证

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，超级加密

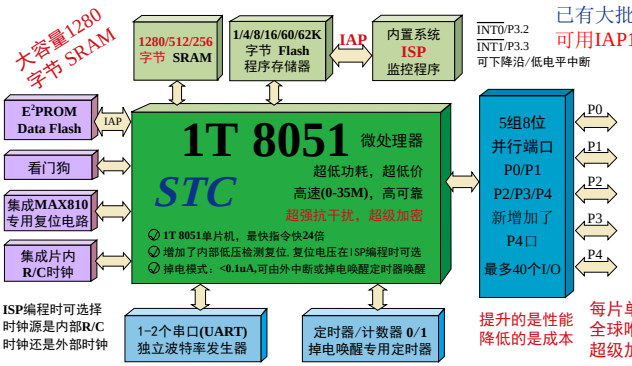
8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

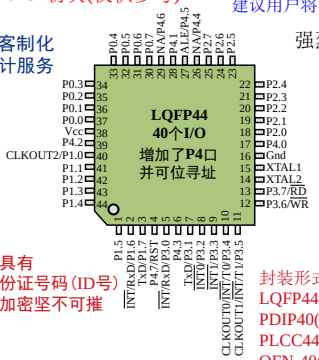
STC11/10xx系列带外部数据总线的 1T 8051 单片机，高速，高可靠，1-2个串口

传统8051单片机划时代升级换代产品，管脚完全兼容，直接取代传统89C51/89S51系列单片机
已有大批量供货6年以上的历史
可用IAP15F2K61S2仿真(仅供参考)

串口作主机通信时，可控制串口通信在[RxD/P3.0,TxD/P3.1]和[RxD/P1.6,TxD/P1.7]之间任意切换，实现2组串口
建议用户将串口设在[RxD/P1.6,TxD/P1.7]



提供客制化IC设计服务



强烈推荐选择LQFP44贴片封装
SOP28/16贴片封装

全部175℃
8小时高温烘烤

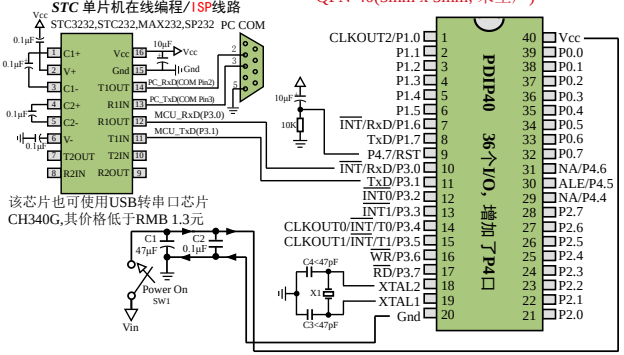
封装形式：
LQFP44(12mm x 12mm, 强烈推荐使用)
PDIP40(正常生产但不推荐使用)
PLCC44(不推荐使用)
QFN-40(5mm x 5mm, 未生产)

提升的是性能 降低的是成本

每片单片机具有全球唯一身份证号码(ID号)
超级加密，加密坚不可摧

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T1外部管脚也能掉电唤醒	独立波特率发生器	掉电唤醒专用定时器	CCP PCA PWM可当外部中断并可掉电唤醒	A/D 8路	看门狗	支持掉电唤醒外部中断	内部低压检测中断	内部复位(可选复位门槛电压)	LQFP44 PDIP40 不推荐PLCC44 部分封装价格(RMB ¥) LQFP44	PDIP40	管脚兼容传统落后型号
STC10Fxx系列单片机选型价格一览表，另有STC10L系列(工作电压2.1V-3.6V)																		
STC10F04	5.5-3.8/3.3	4K	256	-	1-2	无	2	有	-	无	无	有	5个	有	有	¥2.5	¥3.0	89C51
STC10F04XE	5.5-3.8/3.3	4K	512	5K	1-2	无	2	有	-	无	无	有	5个	有	有	¥2.7	¥3.0	89C51
STC10F08	5.5-3.8/3.3	8K	256	-	1-2	无	2	有	-	无	无	有	5个	有	有	¥3.1	¥3.4	89C52
STC10F08XE	5.5-3.8/3.3	8K	512	5K	1-2	无	2	有	-	无	无	有	5个	有	有	¥3.2	¥3.5	89C52
STC10F10XE	5.5-3.8/3.3	10K	512	3K	1-2	无	2	有	-	无	无	有	5个	有	有	¥3.6	¥3.99	89C54
STC10F12XE	5.5-3.8/3.3	12K	512	1K	1-2	无	2	有	-	无	无	有	5个	有	有	¥3.99	¥4.2	89C54
IAP10F14X	5.5-3.8	14K	512	IAP	1-2	无	2	有	-	无	无	有	5个	有	有	¥4.2	¥4.3	用户可将用户程序区的程序Flash当EEPROM用
STC11Fxx系列单片机选型价格一览表，另有STC11L系列(工作电压2.1V-3.6V)																		
STC11F16XE	5.5-4.1/3.7	16K	1280	45K	1-2	无	2	有	有	无	无	有	5个	有	有	¥4.3	¥4.5	89C54
STC11F32XE	5.5-4.1/3.7	32K	1280	29K	1-2	无	2	有	有	无	无	有	5个	有	有	¥4.6	¥4.8	89C58
STC11F56XE	5.5-4.1/3.7	56K	1280	5K	1-2	无	2	有	有	无	无	有	5个	有	有	¥4.7	¥4.9	89C516
STC11F60XE	5.5-4.1/3.7	60K	1280	1K	1-2	无	2	有	有	无	无	有	5个	有	有	¥4.7	¥4.9	89C516
IAP11F62X	5.5-4.1/3.7	62K	1280	IAP	1-2	无	2	有	有	无	无	有	5个	有	有	¥4.7	¥4.9	用户可将用户程序区的程序Flash当EEPROM用

以上单价为200K/M起定量，以上价格运费由客户承担，**零售10片起，如对价格不满，可来电要求降价**



大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无...

复位脚：烧录程序时如设置为I/O口，可当I/O口使用或浮空
不用的I/O口：浮空即可

- 如果I/O口不够用可以用3根普通I/O口线外接74HC595(¥0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D作按键扫描来节省I/O口

宏晶•STC11/10xx系列主要性能：

- 高速：1个时钟/机器周期，增强型8051内核，速度比普通8051快6～12倍
- 宽电压：5.5～4.1V/3.7V，2.1/2.4～3.6V(STC11/10L系列)
- 低功耗设计：**空闲模式(可由任意一个中断唤醒)**
- 低功耗设计：掉电模式**(可由任意一个外部中断唤醒，可支持下降沿/低电平和远程**唤醒，STC11xx系列还可通过内部掉电唤醒专用定时器唤醒)
- 支持掉电唤醒的管脚：P3.2/INT0,P3.3/INT1,T0/P3.4,T1/P3.5,RxD/P3.0(或RxD/P1.6)
- 内部掉电唤醒专用定时器(只有STC11系列才有，STC10系列无)
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部RC振荡器可选，在ISP下载编程用户程序时设置
- 4/8/12/16/32/48/60/62K字节片内Flash程序存储器，擦写次数10万次以上
- 1280/512/256字节片内RAM数据存储器
- 大容量片内EEPROM功能, 擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程, 无需编程器/仿真器
- 2个16位定时器，兼容普通8051定时器T0/T1
- 1个独立波特率发生器(故无需T2做波特率发生器)，缺省是T1做波特率发生器
- 可编程时钟输出功能，T0在P3.4输出时钟，T1在P3.5输出时钟，BRT在P1.0输出时钟
- 硬件看门狗（WDT）
- 全双工异步串行口(UART)，兼容普通8051，可当2组串口使用(串口可在P3与P1之间任意切换)
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(36/40个)，复位后为：准双向口/弱上拉(普通8051传统I/O口)
  可设置成四种模式：准双向口/弱上拉，**强推挽/ 强上拉，仅为输入/高阻，开漏**
  每个I/O口驱动能力均可达到20mA，44/40管脚的IC建议整个芯片不要超过120mA

选择宏晶•STC11/10xx系列单片机理由：

★ 超级加密，采用宏晶第六代加密技术
★ 超强抗干扰，超强抗静电，整机轻松过2万伏静电测试
★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
  ----出口欧美的有力保证
★ 输入/输出口多，最多有40个I/O，复位脚如当I/O口使用，可省去外部复位电路
★ 超低功耗：
  掉电模式：**外部中断唤醒功耗** <0.1uA，支持下降沿/低电平和远程唤醒
  STC11xx系列增加了掉电唤醒专用定时器，启动掉电唤醒定时器典型功耗<2μA
  适用于电池供电系统，如水表、气表、便携设备等。
  空闲模式：**典型功耗**<1.3mA,
  正常工作模式：2mA -7mA
★ 在系统可编程，无需编程器，无需仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(**人民币100元)，1万片/人/天**，有自动烧录机接口
★ 内部集成高可靠复位电路，复位脚设置为I/O口使用时，复位脚可浮空

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，超级加密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

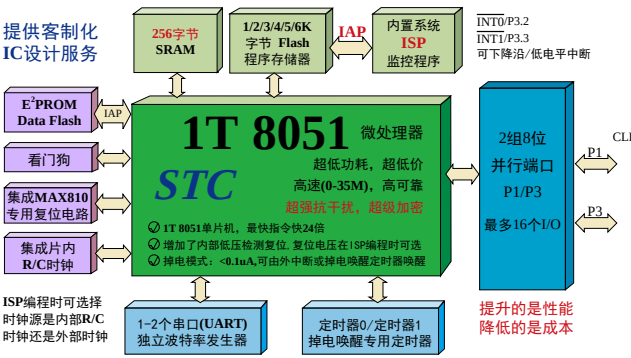
STC11xx系列无外部数据总线 1T 8051 单片机，高速，高可靠，1-2个串口

传统8051单片机划时代升级换代产品，管脚完全兼容，直接取代传统89C51/89S51系列单片机

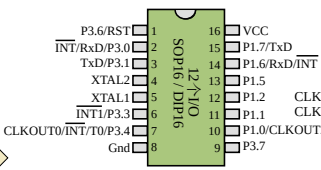
全球最低价：宏晶• STC11F01-SOP16 RMB 1.99元，STC11F04 RMB 2.49元
宏晶• STC11F02-SOP20 RMB 2.20元，可取代89C2051/89C4051

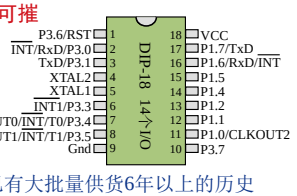
每片单片机具有全球唯一身份证号码(ID号)
超级加密，加密坚不可摧

提供客制化
IC设计服务



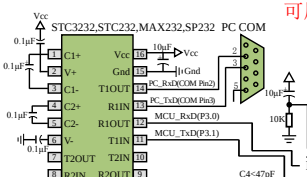
提升的是性能
降低的是成本

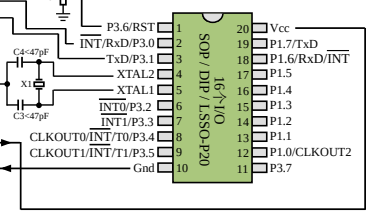




已有大批量供货6年以上的历史
可用IAP15F2K61S2仿真(仅供参考)

封装形式：
SOP16/DIP16
SOP20/DIP20/TSSOP20
DIP-18





STC 单片机在线编程线路
(也可使用USB转串口芯片 CH340G,价格低于RMB 1.3元)

全部175℃
8小时高温烘烤

强烈推荐选择SOP16/20贴片封装

串口作主机通信时，可控制串口通信在[RxD/P3.0,TxD/P3.1]和[RxD/P1.6,TxD/P1.7]之间任意切换，实现2组串口
建议用户将串口设在[RxD/P1.6,TxD/P1.7]

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口并可掉电唤醒	SPI	普通定时器 T0/T1 外部管脚也能掉电唤醒	独立波特率发生器	掉电唤醒专用定时器	CCP PCA PWM 可当外部中断并可掉电唤醒	A/D 8路	看门狗	支持掉电唤醒外部中断	内部低压检测中断	内部复位(可选复位门槛电压)	SOP20/DIP20/TSSOP20 DIP-18 SOP16/DIP16 部分封装价格(RMB ¥) SOP20	DIP20	TSSOP20	DIP18	SOP16	DIP16
STC11Fxx系列单片机选型价格一览表																					
STC11F01E	5.5-4.1/3.5	1K	256	2K	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.1	¥2.3	¥2.3	¥2.3	¥2.0	¥2.1
STC11F02E	5.5-4.1/3.5	2K	256	2K	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.2	¥2.4	¥2.4	¥2.4	¥2.1	¥2.2
STC11F03E	5.5-4.1/3.5	3K	256	2K	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.3	¥2.5	¥2.5	¥2.5	¥2.2	¥2.3
STC11F04E	5.5-4.1/3.5	4K	256	1K	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.4	¥2.6	¥2.6	¥2.6	¥2.3	¥2.4
STC11F05E	5.5-4.1/3.5	5K	256	1K	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.4	¥2.6	¥2.6	¥2.6	¥2.3	¥2.4
IAP11F06	5.5-4.1/3.5	6K	256	IAP	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.4 用户可将用户程序区的程序Flash当EEPROM用	¥2.6	¥2.6	¥2.6	¥2.3	¥2.4
STC11Lxx系列单片机选型价格一览表																					
STC11L01E	3.6-2.4/2.1	1K	256	2K	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.1		¥2.3		¥2.0	
STC11L02E	3.6-2.4/2.1	2K	256	2K	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.2		¥2.4	¥2.4	¥2.1	
STC11L03E	3.6-2.4/2.1	3K	256	2K	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.3		¥2.5		¥2.2	
STC11L04E	3.6-2.4/2.1	4K	256	1K	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.4		¥2.6		¥2.3	
STC11L05E	3.6-2.4/2.1	5K	256	1K	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.4		¥2.6		¥2.3	
IAP11L06	3.6-2.4/2.1	6K	256	IAP	1-2	无	2	-	有	无	无	有	5个	有	有	¥2.4 用户可将用户程序区的程序Flash当EEPROM用	¥2.6	¥2.6	¥2.6	¥2.3	

以上单价为200K/M起定量，以上价格运费由客户承担，**零售10片起，如对价格不满，可来电要求降价**

大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无. . .

复位脚：烧录程序时如设置为I/O口, 可当I/O口使用或浮空
不用的I/O口：浮空即可

宏晶•STC11xx系列主要性能：

- 高速：1个时钟/机器周期，增强型8051内核，速度比普通8051快6～12倍
- 宽电压：5.5～4.1V/3.7V，2.1/2.4～3.6V(STC11L系列)
- 低功耗设计：**空闲模式(可由任意一个中断唤醒)**
- 低功耗设计：掉电模式**(可由任意一个外部中断唤醒，可支持下降沿/低电平和**远程唤醒，STC11xx系列还可通过内部掉电唤醒专用定时器唤醒)
- 支持掉电唤醒的管脚：P3.2/INT0,P3.3/INT1,T0/P3.4,T1/P3.5,RxD/P3.0(或RxD/P1.6)
- 内部掉电唤醒专用定时器(只有STC11系列才有，STC10系列无)
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部RC振荡器可选，在ISP下载编程用户程序时设置
- 1/2/3/4/5/6K字节片内Flash程序存储器，擦写次数10万次以上
- 256字节片内RAM数据存储器
- 大容量片内EEPROM功能, 擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程, 无需编程器/仿真器
- 2个16位定时器，兼容普通8051定时器T0/T1
- 1个独立波特率发生器(故无需T2做波特率发生器)，缺省是T1做波特率发生器
- 可编程时钟输出功能，T0在P3.4输出时钟，T1在P3.5输出时钟
- 硬件看门狗（WDT）
- 全双工异步串行口(UART)，兼容普通8051，可当2组串口使用(串口可在P3与P1之间任意切换)
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令

- 通用I/O口(16/14/12个)，复位后为：准双向口/弱上拉(普通8051传统I/O口)
  可设置成四种模式：准双向口/弱上拉，**强推挽/ 强上拉，仅为输入/高阻，开漏**
  每个I/O口驱动能力均可达到20mA, 20/18/16管脚的IC建议整个芯片不要超过70mA
- 如果I/O口不够用可以用3根普通I/O口线外接74HC595(¥0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D作按键扫描来节省I/O口

选择宏晶•STC11xx系列单片机理由：

★ 超级加密，采用宏晶第六代加密技术
★ 超强抗干扰，超强抗静电，整机轻松过2万伏静电测试
★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
　----出口欧美的有力保证
★ 输入/输出口多，最多有16个I/O，复位脚如当I/O口使用，可省去外部复位电路
★ 超低功耗：
　掉电模式：**外部中断唤醒功耗** <0.1uA，支持下降沿/低电平和远程唤醒
　STC11xx系列增加了掉电唤醒专用定时器，启动掉电唤醒定时器典型功耗<2μA
　　适用于电池供电系统，如水表、气表、便携设备等。
　空闲模式：**典型功耗**<1.3mA,
　正常工作模式：2mA -7mA
★ 在系统可编程, 无需编程器, 无需仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(**人民币100元**)，**1万片/人/天**，有自动烧录机接口
★ 内部集成高可靠复位电路, 复位脚设置为I/O口使用时，复位脚可浮空

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

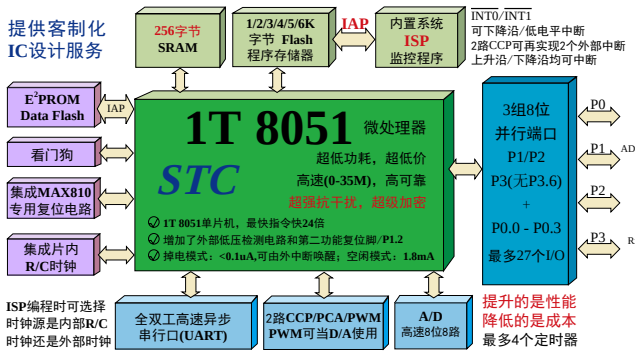
STC micro™ 宏晶科技

超强抗干扰，超级加密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

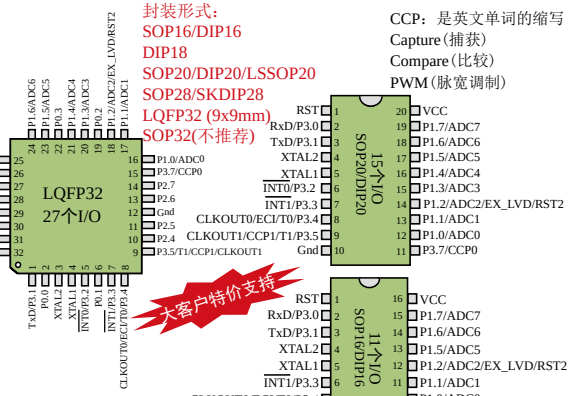
官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

STC12C5201AD系列 1T 8051 单片机，超低价，2路CCP/PCA/PWM，高速A/D



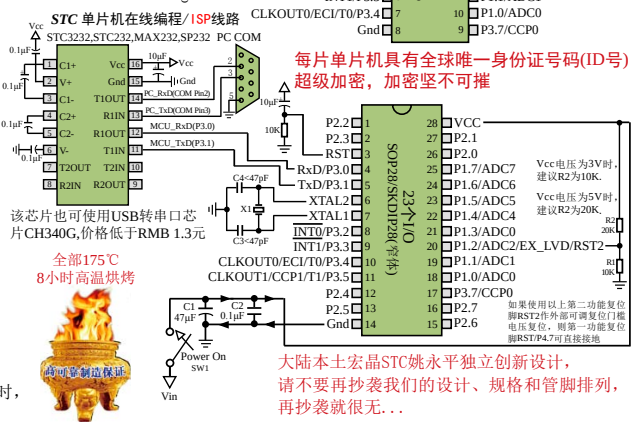
封装形式：
SOP16/DIP16
DIP18
SOP20/DIP20/LSSOP20
SOP28/SKDIP28
LQFP32 (9x9mm)
SOP32(不推荐)

CCP：是英文单词的缩写
Capture(捕获)
Compare(比较)
PWM(脉宽调制)



型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	DPTR	串行口	SPI	普通定时器	CCP PCA PWM可当外部中断并可掉电唤醒	A/D 8路 30万次每秒(2路PWM可当2路D/A使用)	外部中断4路,低压检测中断	外部复位可调门槛电压	LQFP32/SOP32(不推荐) SOP28/SKDIP28 SOP20/DIP20/TSSOP20 SOP16/DIP16 部分封装价格(RMB ¥)								
								T0/T1外部管脚也能掉电唤醒					SOP16价格	DIP16价格	SOP20价格	DIP20价格	TSSOP20价格	SOP28价格	SKDIP28价格	LQFP32价格	SOP32价格
STC12C5201AD系列单片机选型价格一览表 特别提醒：2路CCP/PCA/PWM还可当2路定时器使用，另有STC12LE系列(工作电压2.1V-3.6V)																					
STC12C5201AD	5.5-4.0	1K	256	2K	1	无	2	2-ch	1	8位	有	有	¥2.3	¥2.4	¥2.4	¥2.6	¥2.6				
STC12C5202AD	5.5-4.0	2K	256	2K	1	无	2	2-ch	1	8位	有	有	¥2.4	¥2.5	¥2.5	¥2.7	¥2.7	¥2.8	¥3.0	¥3.0	¥3.1
STC12C5203AD	5.5-4.0	3K	256	2K	1	无	2	2-ch	1	8位	有	有	¥2.5	¥2.6	¥2.6	¥2.8	¥2.8	¥2.8	¥3.0	¥3.1	¥3.2
STC12C5204AD	5.5-4.0	4K	256	1K	1	无	2	2-ch	1	8位	有	有	¥2.6	¥2.7	¥2.7	¥2.9	¥2.9	¥2.8	¥3.0	¥3.1	¥3.2
STC12C5205AD	5.5-4.0	5K	256	1K	1	无	2	2-ch	1	8位	有	有	¥2.6	¥2.7	¥2.7	¥2.9	¥2.9	¥2.8	¥3.0	¥3.1	¥3.2
STC12C5206AD	5.5-4.0	6K	256	IAP	1	无	2	2-ch	1	8位	有	有	¥2.6	¥2.7	¥2.7	¥2.9	¥2.9	¥2.8	¥3.0	¥3.1	¥3.2
用户可将用户程序区的程序Flash当EEPROM使用																					

以上单价为200K/M起定量，以上价格运费由客户承担，**零售10片起，如对价格不满，可来电要求降价**



宏晶•STC12C5201AD系列主要性能：

- 高速：1个时钟/机器周期，增强型8051内核，速度比普通8051快6～12倍
- 宽电压：5.5～4.0V，2.2～3.6V(STC12LE5201AD系列)
- 增加第二复位功能脚/P1.2(高可靠复位，可调整复位门槛电压，频率<12MHz时，无需此功能)
- 增加外部掉电检测电路/P1.2，可在掉电时，及时将数据保存进EEPROM，正常工作时无需操作EEPROM
- 低功耗设计：**空闲模式(可由任意一个中断唤醒)，掉电模式(可由外部中断唤醒)**
- 支持掉电唤醒的管脚：P3.2/$\overline{INT0}$, P3.3/$\overline{INT1}$, T0/P3.4, T1/P3.5, RxD/P3.0, P3.7/CCP0, P3.5/CCP1, EX_LVD/P1.2
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部RC振荡器可选，在ISP下载编程用户程序时设置
- 1K/2K/3K/4K/5K/6K字节片内Flash程序存储器，擦写次数10万次以上
- 256字节片内RAM数据存储器
- 片内EEPROM功能，擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- 8通道，8位高速ADC，速度可达30万次/秒，2路PWM还可当2路D/A使用
- 2通道捕获/比较单元(CCP/PCA/PWM)，
  ---也可用来再实现2个定时器或2个外部中断(支持上升沿/下降沿中断)
- 2个16位定时器，兼容普通8051的定时器T0/T1, 2路PCA可再实现2个定时器
- 可编程时钟输出功能，T0在P3.4输出时钟，T1在P3.5输出时钟
- 硬件看门狗（WDT）
- 全双工异步串行口(UART)，兼容普通8051的串口
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(27/23/15个)，复位后为：准双向口/弱上拉（普通8051传统I/O口）
  可设置成四种模式：准双向口/弱上拉，**强推挽/强上拉，仅为输入/高阻，开漏**
  每个I/O口驱动能力均可达到20mA，但建议整个芯片不要超过90mA
- 如选32-Pin，推荐优先选择LQFP32

复位脚：烧录程序时如设置为I/O口, 可当I/O口使用或浮空
EX_LVD：是外部低压检测中断/比较器
不用的I/O口：浮空即可

可用IAP15F2K61S2仿真(仅供参考)

- 如果I/O口不够用可以用3根普通I/O口线外接74HC595(¥0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D作按键扫描来节省I/O口

选择宏晶•STC12C5201AD系列单片机理由：

★ 超级加密，采用宏晶第六代加密技术
★ 超强抗干扰：
1. 高抗静电(ESD保护)，整机轻松过2万伏静电测试
2. 轻松过4KV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动
4. 宽温度范围，-40℃～+85℃

★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
----出口欧美的有力保证
★ 超低功耗：
1. **掉电模式:外部中断唤醒功耗** <0.1uA
2. 空闲模式：典型功耗 1.8mA,
3. 正常工作模式:2.7mA -7mA
4. **掉电模式可由外部中断唤醒，适用于电池供电系统，如水表、气表、便携设备等**

★ 在系统可编程,无需编程器,无需仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元),1万片/人/天，有自动烧录机接口
★ 内部集成高可靠复位电路，外部复位电路可彻底省掉，当然也可以继续用外部复位电路
★ 全部175℃，8小时高温烘烤，高可靠制造保证

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™

宏 晶 科 技

超强抗干扰，超级加密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com　www.GXWMCU.com
南通 Tel: 0513-5501 2928　5501 2929
深圳 Tel: 0755-8294 8411　8294 8412

STC12C5620AD系列 1T 8051 单片机，4路PWM/PCA，8路10位A/D转换

超小封装TSSOP20, 6.4mm×6.4mm

封装形式：
SOP20/DIP20/TSSOP20
SOP28/SKDIP28
LQFP32 (9x9mm)
SOP32(不推荐)

强烈推荐选择SOP20/28,
LQFP32贴片封装

传统插件DIP
封装稳定供货

每片单片机具有
全球唯一身份证号码(ID号)
超级加密，加密坚不可摧

STC12C5620AD
系列可直接取代
传统12C5410/
2052AD系列

全部175℃
8小时高温烘烤

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口并可掉电唤醒	SPI	普通定时器计数器T0/T1外部管脚也能掉电唤醒	PCA PWM可当外部中断并可掉电唤醒	DPTR	A/D 8路10万次每秒(4路PWM可当4路D/A使用)	看门狗	内部低压检测中断	内部复位(可选复位门槛电压)	SOP20价格	DIP20价格	TSSOP20价格	SOP28价格	SKDIP28价格	LQFP32价格	SOP32价格
														LQFP32/SOP32(不推荐) SOP28/SKDIP28 SOP20/DIP20/TSSOP20 部分封装价格(RMB ¥)						
STC12C5620AD系列单片机选型价格一览表 特别提醒：4路CCP/PCA/PWM还可当4路定时器使用，另有STC12LE系列(工作电压2.2V-3.6V)																				
STC12C5604AD	5.5-3.5	4K	768	4K	1	有	2	4-ch	1	10位	有	有	有	¥3.60起						
STC12C5608AD	5.5-3.5	8K	768	4K	1	有	2	4-ch	1	10位	有	有	有				¥3.8起			
STC12C5612AD	5.5-3.5	12K	768	IAP	1	有	2	4-ch	1	10位	有	有	有	用户可将用户程序区的程序Flash当EEPROM使用						
STC12C5616AD	5.5-3.5	16K	768	4K	1	有	2	4-ch	1	10位	有	有	有						¥4.99起	
STC12C5620AD	5.5-3.5	20K	768	4K	1	有	2	4-ch	1	10位	有	有	有							
STC12C5624AD	5.5-3.5	24K	768	4K	1	有	2	4-ch	1	10位	有	有	有							
STC12C5628AD	5.5-3.5	28K	768	IAP	1	有	2	4-ch	1	10位	有	有	有	用户可将用户程序区的程序Flash当EEPROM使用						
STC12C5630AD	5.5-3.5	30K	768	IAP	1	有	2	4-ch	1	10位	有	有	有	用户可将用户程序区的程序Flash当EEPROM使用						

以上单价为200K/M起定量，以上价格运费由客户承担，零售10片起，**如对价格不满，可来电要求降价**

STC 单片机在线编程/ISP线路
(也可使用USB转串口芯片
CH340G,价格低于RMB 1.3元)

大客户特价支持

提供客制化IC设计服务

可用IAP15F2K61S2仿真(仅供参考)

宏晶•STC12C5620AD系列主要性能：

- 高速：1个时钟/机器周期，增强型8051内核，速度比普通8051快8～12倍
- 宽电压：5.5～3.5V，2.2～3.6V(STC12LE5620AD系列)
- 低功耗设计：空闲模式(可由任意一个中断唤醒)，掉电模式(可由外部中断唤醒)
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部RC振荡器可选，在ISP下载编程用户程序时设置
- 30K/28K/24K/20K/16K/12K/8K/4K节片内Flash程序存储器，擦写次数10万次以上
- 768字节(256+512)片内RAM数据存储器
- 片内EEPROM功能，**擦写次数**10万次以上
- ISP/IAP，在系统可编程/在应用可编程,无需编程器/仿真器
- 8通道，10位ADC，速度可达10万次/秒，4路PWM还可当4路D/A使用
- 4通道捕获/比较单元(PWM/PCA)，
  ---也可用来再实现4个定时器或4个外部中断(支持上升沿/下降沿中断)
- 2个16位定时器，兼容普通8051的定时器T0/T1，4路PCA可再实现4个定时器
- 可编程时钟输出功能，T0在P1.0输出时钟，T1在P1.1输出时钟
- 硬件看门狗（WDT）
- SPI高速同步串行通信端口
- 全双工异步串行口(UART)，兼容普通8051的串口
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(27/23/15个)，复位后为：　准双向口/弱上拉（普通8051传统I/O口）
  可设置成四种模式：准双向口/弱上拉，**强推挽/强上拉，仅为输入/高阻，开漏**
  每个I/O口驱动能力均可达到20mA，但建议整个芯片不要超过90mA

★ 如选32-Pin，推荐选LQFP32
★ 如果I/O口不够用，可以用3根普通I/O口线外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D做按键扫描来节省I/O口

大陆本土宏晶STC姚永平独立创新设计，请不要再抄袭
我们的设计、规格和管脚排列，再抄袭就很无...

选择宏晶•STC12C5620AD系列单片机理由：

★ 超级加密，采用宏晶第六代加密技术
★ 超强抗干扰：
　1. 高抗静电(ESD保护),整机轻松过2万伏静电测试
　2. 轻松过4KV快速脉冲干扰(EFT测试)
　3. 宽电压，不怕电源抖动　　　4. 宽温度范围，-40℃~+85℃
★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
　----出口欧美的有力保证
★ 超低功耗：
　1. **掉电模式:外部中断唤醒功耗** <0.1uA
　2. 空闲模式：典型功耗 1.8mA,　　　3. 正常工作模式:2.7mA -7mA
　4. **掉电模式可由外部中断唤醒**，适用于电池供电系统，如水表、气表、便携设备等
★ 在系统可编程,无需编程器,无需仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元),1万片/人/天，有自动烧录机接口
★ 内部集成专用**复位电路，有2级复位门槛电压可选**，12MHz以下可放心使用内部复位，**外部复位电路可保留，也可以不用(复位脚直接接地)**

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

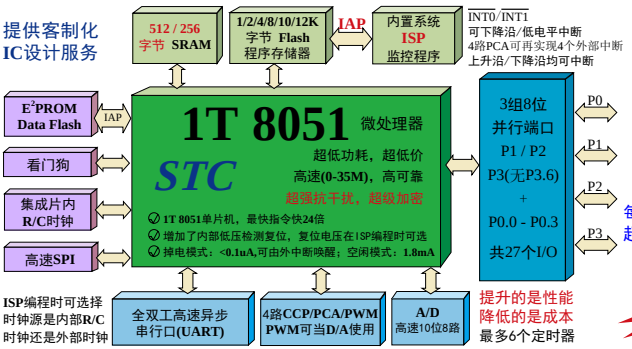
STC micro™ 宏晶科技

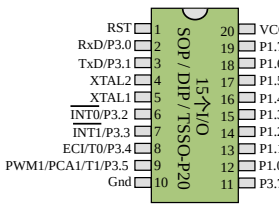
超强抗干扰，超级加密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC12C5410AD/2052AD系列 1T 8051 单片机，4路PWM/PCA，8路10位A/D



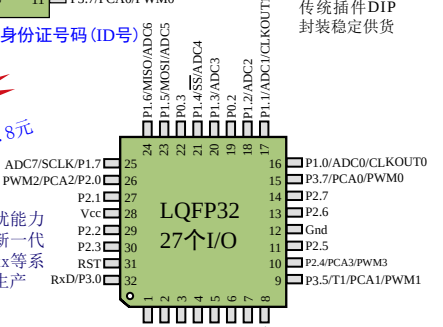


封装形式：
SOP20/DIP20/TSSOP20
SOP28/SKDIP28
LQFP32 (9x9mm)
SOP32(不推荐)
超小封装TSSOP20, 6.4mm x 6.4mm
强烈推荐选择SOP20/28,
LQFP32贴片封装

传统插件DIP
封装稳定供货

每片单片机具有全球唯一身份证号码(ID号)
超级加密，加密坚不可摧

大客户特价支持
STC12C1052, 人民币3.8元



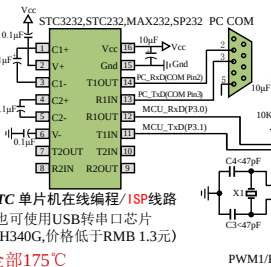
尽量优先选择成本更低，抗干扰能力更强，采用新加密技术的宏晶新一代STC15xx/11xx/12C52xx/12C56xx等系列取代，原有老产品继续长期生产

型号	工作电压(V)	Flash程序存储器(byte)	SRAM字节	EEPROM	串行口并可掉电唤醒	SPI	普通定时器计数器	PCA PWM可当外部中断并可掉电唤醒	DPTR	A/D 8路10万次每秒(4路PWM可当4路D/A使用)	看门狗	内部复位(可选复位门槛电压)	LQFP32/SOP32(不推荐) SOP28/SKDIP28 SOP20/DIP20/TSSOP20 部分封装价格(RMB ¥) SOP20价格(RMB ¥)	SOP28价格(RMB ¥)	LQFP32价格(RMB ¥)
STC12C5410AD系列单片机选型价格一览表 特别提醒：4路CCP/PCA/PWM还可当4路定时器使用，另有STC12LE系列(工作电压2.2V-3.6V)															
STC12C5401AD	5.5-3.5	1K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5402AD	5.5-3.5	2K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5404AD	5.5-3.5	4K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5406AD	5.5-3.5	6K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5408AD	5.5-3.5	8K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5410AD	5.5-3.5	10K	512	有	1	有	2	4-ch	1	10位	有	有			
STC12C5412AD	5.5-3.5	12K	512	IAP	1	有	2	4-ch	1	10位	有	有	用户可将用户程序区的程序Flash当EEPROM使用		
STC12C2052AD系列单片机选型价格一览表 特别提醒：2路CCP/PCA/PWM还可当2路定时器使用，另有STC12LE系列(工作电压2.2V-3.6V)															
STC12C1052AD	5.5-3.5	1K	256	有	1	有	2	2-ch	1	8位	有	有	SOP20/DIP20/TSSOP20		
STC12C2052AD	5.5-3.5	2K	256	有	1	有	2	2-ch	1	8位	有	有	SOP20/DIP20/TSSOP20		
STC12C4052AD	5.5-3.5	4K	256	有	1	有	2	2-ch	1	8位	有	有	SOP20/DIP20/TSSOP20		
STC12C5052AD	5.5-3.5	5K	256	IAP	1	有	2	2-ch	1	8位	有	有	用户可将用户程序区的程序Flash当EEPROM使用		

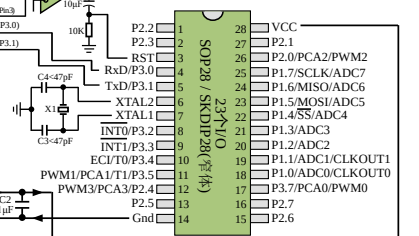
以上单价为200K/M起定量，以上价格运费由客户承担，**零售10片起，如对价格不满，可来电要求降价**

封装为SKDIP28的单片机比封装为SOP28的单片机要贵0.2元；
封装为DIP20的单片机比封装为SOP20的单片机要贵0.2元。

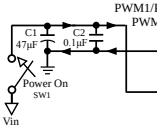
可用IAP15F2K61S2仿真(仅供参考)



STC 单片机在线编程/ISP线路
(也可使用USB转串口芯片CH340G,价格低于RMB 1.3元)



全部175℃
8小时高温烘烤



大陆本土宏晶STC姚永平独立创新设计，请不要再抄袭我们的设计、规格和管脚排列，再抄袭就很无. . .

★ 如选32-Pin，推荐选LQFP32
★ 如果I/O口不够用，可以用3根普通I/O口线外接74HC595(参考价0.21元)来扩展I/O口，并可多芯片级联扩展几十个I/O口，还可用A/D做按键扫描来节省I/O口

宏晶•STC12C5410AD/2052AD系列主要性能：

- 高速：1个时钟/机器周期，增强型8051内核，速度比普通8051快6～12倍
- 宽电压：5.5～3.5V，2.2～3.8V(STC12LE5410AD/2052AD系列)
- 低功耗设计：**空闲模式(可由任意一个中断唤醒)，掉电模式(可由外部中断唤醒)**
- 工作频率：0～35MHz，相当于普通8051：0～420MHz
- 时钟：外部晶体或内部RC振荡器可选，在ISP下载编程用户程序时设置
- 16K/12K/10K/8K/6K/4K/2K/1K节片内Flash程序存储器，擦写次数10万次以上
- 512 / 256 字节**片内**RAM数据存储器
- 片内EEPROM功能，擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程，无需编程器/仿真器
- 8通道，10位ADC，STC12C2052AD系列为8位ADC，4路PWM还可当4路D/A使用
- 4通道捕获/比较单元(PWM/PCA)，STC12C2052AD系列为2通道
  ---也可用来再实现4个定时器或4个外部中断(支持上升沿/下降沿中断)
- 2个16位定时器，兼容普通8051的定时器T0/T1，4路PCA可再实现4个定时器
- 可编程时钟输出功能，T0在P1.0输出时钟，T1在P1.1输出时钟
- 硬件看门狗(WDT)
- SPI高速**同步串行通信端口**
- 全双工异步串行口(UART)，兼容普通8051的串口
- 先进的指令集结构，兼容普通8051指令集
  4组8个8位**通用工作寄存器(共32个通用寄存器)**
  有硬件乘法/除法指令
- 通用I/O口(27/23/15个)，复位后为：准双向口/弱上拉(普通8051传统I/O口)
  可设置成四种模式：准双向口/弱上拉，**强推挽/强上拉，仅为输入/高阻，开漏**
  每个I/O口驱动能力均可达到20mA，但建议整个芯片不要超过90mA

选择宏晶•STC12C5410AD/2052AD系列单片机理由：

★ 超级加密
★ 超强抗干扰：
1. 高抗静电(ESD保护)　2. 轻松过4KV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动　4. 宽温度范围，-40℃~+85℃
★ 速度快，1个时钟/机器周期，可用低频晶振，大幅降低EMI
----出口欧美的有力保证
★ 超低功耗：
1. **掉电模式:外部中断唤醒功耗** <0.1uA
2. 空闲模式：典型功耗 1.8mA，　3. 正常工作模式:2.7mA -7mA
4. **掉电模式可由外部中断唤醒，适用于电池供电系统，如水表、气表、便携设备等**
★ 所有封装均符合欧盟RoHS要求
★ 在系统可编程，无需编程器，无需仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元)，1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，加密性强

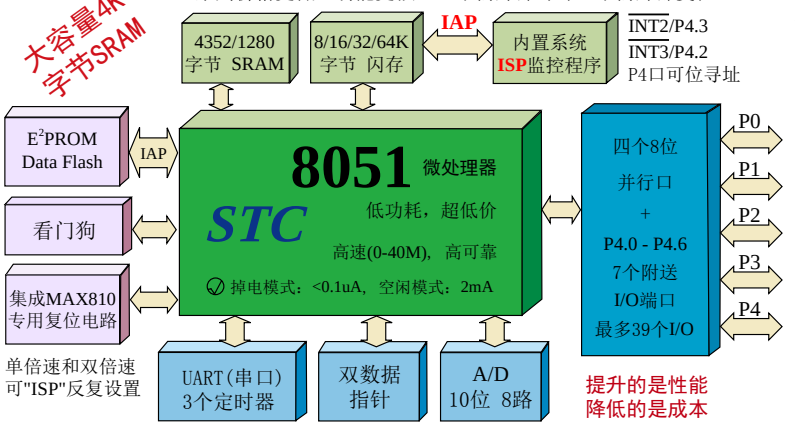
8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

宏晶·STC 89C51 90C51 系列单片机，直接兼容传统8051

欢迎使用STC90C51系列(软硬件完全兼容)取代89C51系列，STC90C51系列抗干扰能力更强，成本更低
或请使用第六代加密技术加密、加密性强的STC11/10xx系列取代89C51系列
或强烈建议使用第九代加密技术加密、无法解密的STC15xx系列取代89C51系列，
STC15系列价格更低、功能更强，且不需外部时钟、不需外部复位

已有大批量供货10年以上的历史
可用IAP15F2K61S2仿真(仅供参考)



选择宏晶·STC90C51系列单片机理由：

★ 超大容量SRAM，最高达4.2K字节SRAM
★ 直接取代传统8051单片机，成本更低
★ 10位A/D，可达25万次/秒
★ 超强抗干扰：
1. 高抗静电(ESD保护，整机轻松过2万伏静电测试)
2. 轻松过4KV快速脉冲干扰(EFT测试)
3. 宽电压，不怕电源抖动
4. 宽温度范围，-40℃~+85℃
★ 三大降低单片机时钟对外部电磁辐射的措施：
----出口欧美的有力保证
1. 禁止ALE输出；
2. 如选6时钟/机器周期，外部时钟频率可降低一半；
3. 单片机时钟振荡器增益可设为1/2gain
★ 超低功耗：
1. 掉电模式：**典型功耗功耗** <0.1uA
2. 正常工作模式：4mA -7mA
3. **掉电模式可由外部中断唤醒，适用于电池供电系统，如水表、气表、便携设备等。**
★ 在系统可编程，无需编程器**及仿真器，可远程升级**
★ 送USB型联机/**脱机下载烧录工具STC-U7(人民币100元)**，1万片/人/天
★ 全部175℃，8小时高温烘烤，高可靠制造保证

型号	工作电压(V)	Flash程序存储器(字节)	SRAM字节	EEPROM(字节)	定时器	A/D 10位8路	降低EMI	双倍速	I/O口	支持掉电唤醒外部中断	内置复位	看门狗	ISP	IAP	LQFP44价格	PIDP40价格	PLCC44价格	兼容传统落后型号	价格更低 功能更强 不需外部时钟 不需外部复位的替代型号
STC89C51RC	5.5-3.5	4K	512	9K	3个	-	√	√	39个	4个	有	有	√	√	¥2.8	¥3.3	¥3.4	89C51	
STC90C51RC	5.5-3.5	4K	512	9K	3个	-	√	√	39个	4个	有	有	√	√	¥2.7	¥3.2	¥3.3	89C51	STC15W404S
STC15W404S	5.5-2.4	4K	512	9K	3个	-	√		42个	5个	强	强	√	√	¥2.5	¥3.0			
STC89C52RC	5.5-3.5	8K	512	5K	3个	-	√	√	39个	4个	有	有	√	√	¥2.8	¥3.1	¥3.4	89C52	
STC90C52RC	5.5-3.5	8K	512	5K	3个	-	√	√	39个	4个	有	有	√	√	¥2.7	¥3.0	¥3.3	89C52	STC15W408S
STC15W408S	5.5-2.4	8K	512	5K	3个	-	√		42个	5个	强	强	√	√	¥2.7	¥3.0			
STC89C53RC	5.5-3.5	12K	512	-	3个	-	√	√	39个	4个	有	有	√	√	¥3.5	¥4.0	¥4.1	89C53	
STC90C53RC	5.5-3.5	12K	512	-	3个	-	√	√	39个	4个	有	有	√	√	¥3.4	¥3.9	¥4.0	89C54	IAP15W413S
IAP15W413S	5.5-2.4	13K	512	IAP	3个	-	√		42个	5个	强	强	√	√	¥2.8	¥3.3			
STC89C54RD+	5.5-3.5	16K	1280	45K	3个	-	√	√	39个	4个	有	有	√	√	¥4.5	¥5.0	¥5.1	89C54	
STC90C54RD+	5.5-3.5	16K	1280	45K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C54	STC15W1K16S
STC15W1K16S	5.5-2.6	16K	1024	13K	3个	-	√		42个	5个	强	强	√	√	¥3.5	¥4.0			
STC89C58RD+	5.5-3.5	32K	1280	29K	3个	-	√	√	39个	4个	有	有	√	√	¥4.5	¥5.0	¥5.1	89C58	
STC90C58RD+	5.5-3.5	32K	1280	29K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C58	IAP15W1K29S
IAP15W1K29S	5.5-2.6	29K	1024	IAP	3个	-	√		42个	5个	强	强	√	√	¥3.8	¥4.3			
STC89C516RD+	5.5-3.5	61K	1280		3个	-	√	√	39个	4个	有	有	√	√	¥4.5	¥5.0	¥5.1	89C516	
STC90C516RD+	5.5-3.5	61K	1280		3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	IAP15F2K61S
IAP15F2K61S	5.5-3.8	61K	2048	IAP	3个	-	√		42个	5个	强	强	√	√	¥4.0	¥4.5			
STC90C510RD+	5.5-3.5	40K	1280	21K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	
STC90C512RD+	5.5-3.5	48K	1280	13K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	
STC90C514RD+	5.5-3.5	56K	1280	5K	3个	-	√	√	39个	4个	有	有	√	√	¥4.4	¥4.9	¥5.0	89C516	
STC90C54AD	5.5-3.8	16K	4352	45K	3个	有	√	√	39个	4个	有	有	√	√	¥7.4	¥7.9	¥8.0	特别要求:A/D输入电压需<1/2Vcc	
STC90C58AD	5.5-3.8	32K	4352	29K	3个	有	√	√	39个	4个	有	有	√	√	¥8.4	¥8.9	¥9.0		
STC90C514AD	5.5-3.8	56K	4352	5K	3个	有	√	√	39个	4个	有	有	√	√	¥9.4	¥9.9	¥10.0		
STC90C516AD	5.5-3.8	61K	4352	-	3个	有	√	√	39个	4个	有	有	√	√	¥10.4	¥10.9	¥11.0		

以上单价为10K/M起定量，以上价格运费由客户承担，**零售1片起，如对价格不满，可来电要求降价**

用"不需外部时钟/外部复位"的"STC15W404S/STC15W408S/IAP15W413S，取代89C51/89C52/89C53，成本更低
用"不需外部时钟/外部复位"的"STC15W1K16S/STC15W1K24S/IAP15W1K29S" 取代89C54/89C58，成本更低
用"不需外部时钟/外部复位"的"IAP15F2K61S" 取代89C54/89C58/89C516，成本更低

宏晶STC90C58AD系列单片机(带A/D转换)的串口可从[P3.0/RxD，P3.1/TxD]切换到[P1.6/RxD，P1.7/TxD]，可当双串口使用

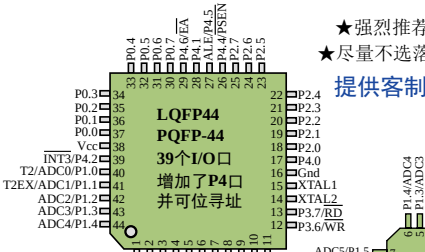
宏晶STC90C58AD系列(带A/D转换)P4口地址在C0H，其它不带A/D转换的89/90系列单片机P4口地址在E8H

免费索取

从网上下载样品申请单，
传真至深圳申请STC单片机
样片及ISP下载/编程工具

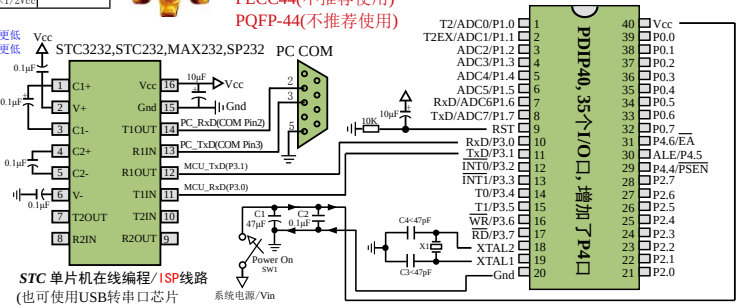
★强烈推荐LQFP44小型封装★
★尽量不选落后的PDIP和PLCC封装★

提供客制化IC设计服务



全部175℃
8小时高温烘烤

封装形式：
PDIP40(正常生产，不推荐使用)
LQFP44(12x12mm，强烈推荐)
PLCC44(不推荐使用)
PQFP-44(不推荐使用)



STC 单片机在线编程/ISP线路
(也可使用USB转串口芯片
CH340G,价格低于RMB 1.3元)

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！

STC micro™ 宏晶科技

超强抗干扰，无法解密

8051单片机全球第一品牌，全球最大的8051单片机设计公司
全部中国大陆本土独立自主知识产权；品质保证：TSMC上海制造

官方网站：www.STCMCU.com www.GXWMCU.com
南通 Tel: 0513-5501 2928 5501 2929
深圳 Tel: 0755-8294 8411 8294 8412

STC15W4K60S4系列1T 8051单片机，4K字节SRAM，超高速四串口，6路15位PWM

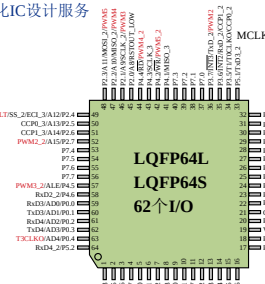
不需外部晶振的单片机
不需外部复位的单片机

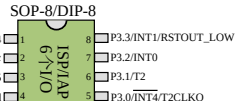
送仿真器

全球第一款真正意义上的单片机
ISP/IAP技术全球领导者

采用宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞

提供客制化IC设计服务



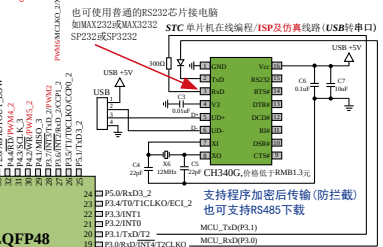


STC15F101W系列, 大批量供货中
RMB1. 2元起 (STC15F101W)

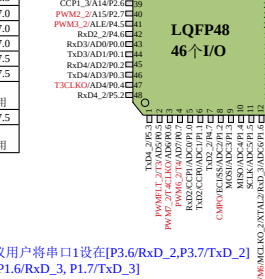
封装形式：
LQFP64L(16mm x 16mm)
LQFP64S(12mm x 12mm)
LQFP48(9mm x 9mm, 强烈推荐)
LQFP44(12mm x 12mm, 强烈推荐)
PDIP40(正常生产，但不推荐)
LQFP32(9mm x 9mm)
SOP28
SKDIP28
强烈推荐选择SOP28, LQFP44/48/64贴片封装

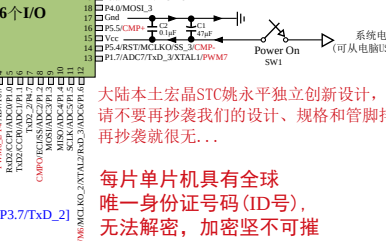
全部175℃ 8小时高温烘烤

也可使用普通的RS232芯片接电脑
如MAX232或MAX3232 SP232或SP3232



支持程序加密后传输 (防拦截)
也可支持RS485下载





大陆本土宏晶STC姚永平独立创新设计，
请不要再抄袭我们的设计、规格和管脚排列，
再抄袭就很无...

每片单片机具有全球
唯一身份证号码(ID号)，
无法解密，加密坚不可摧

型号	工作电压(V)	Flash程序存储器(byte)	大容量SRAM字节	串口并可当9组使用	SPI	普通定时器计数器 T0-T4外部管脚也能掉电唤醒	8路 PWM 6路15位专门的PWM(带死区控制)	2路CCP 10位PWM可当外部中断并可掉电唤醒	掉电唤醒专用定时器	标准外部中断支持掉电唤醒	A/D 8路 2路CCP可当3路D/A使用	比较器(可当1路ADC使用, 可作外部掉电检测)	DPTR	EEPROM	内部低压检测中断并可掉电唤醒	看门狗	内部复位(可选复位门槛电压)	内部高精准时钟(可省外部晶振)	程序加密后传输(防止程序被拦截)	支持RS485下载	所有封装 LQFP64/LQFP48/LQFP44/PDIP40 LQFP32/SOP28/SKDIP28 部分封装价格(RMB ¥) PDIP40	LQFP44	LQFP48	LQFP64
STC15W4K60S4系列单片机选型价格一览表，2014年8月开始供货 特别提醒：8路PWM可当8路D/A使用, 2路CCP可当2个定时器或2个外部中断使用																								
STC15W4K16S4	2.4-5.5	16K	4K	4	4	有	5	6-ch	2-ch	有	有	有	10位	√	2	45K	有	有	16级	有	是	有	是	有	是	¥5.5	¥5.0	¥5.2	¥6.0
STC15W4K32S4	2.4-5.5	32K	4K	4	4	有	5	6-ch	2-ch	有	有	有	10位	√	2	29K	有	有	16级	有	是	有	是	有	是	¥6.0	¥5.5	¥5.7	¥6.5
STC15W4K40S4	2.4-5.5	40K	4K	4	4	有	5	6-ch	2-ch	有	有	有	10位	√	2	21K	有	有	16级	有	是	有	是	有	是	¥6.5	¥6.0	¥6.2	¥7.0
STC15W4K48S4	2.4-5.5	48K	4K	4	4	有	5	6-ch	2-ch	有	有	有	10位	√	2	13K	有	有	16级	有	是	有	是	有	是	¥6.5	¥6.0	¥6.2	¥7.0
STC15W4K56S4	2.4-5.5	56K	4K	4	4	有	5	6-ch	2-ch	有	有	有	10位	√	2	5K	有	有	16级	有	是	有	是	有	是	¥6.5	¥6.0	¥6.2	¥7.0
STC15W4K60S4	2.4-5.5	60K	4K	4	4	有	5	6-ch	2-ch	有	有	有	10位	√	2	1K	有	有	16级	有	是	有	是	有	是	¥7.0	¥6.5	¥6.7	¥7.5
IAP15W4K61S4 本身就是仿真器	2.4-5.5	61K	4K	4	4	有	5	6-ch	2-ch	有	有	有	10位	√	2	IAP	有	有	16级	有	是	有	是	有	是	¥7.0	¥6.5	¥6.7	¥7.5
																										用户可将用户程序区的程序Flash当EEPROM使用			
IRC15W4K63S4 使用外部晶振或内部24MHz时钟	2.4-5.5	63.5K	4K	4	4	有	5	6-ch	2-ch	有	有	有	10位	√	2	IAP	有	有	16级	有	是	无	否			¥7.0	¥6.5	¥6.7	¥7.5
																										用户可将用户程序区的程序Flash当EEPROM使用			

以上单价为10K/M起定量，量小每片需加0.1元，以上价格运费由客户承担，零售10片起，如对价格不满，可来电要求降价

程序加密后传输：程序拥有者产品出厂时将源程序和加密钥匙一起烧录MCU中，以后需要升级软件时，就可将程序加密后再用"发布项目程序"功能，生成一个用户自己界面的只有一个升级按钮的简单易用的升级软件，给最终使用者自己升级，而拦截不到您的原始程序。

IRC15W4K63S4型号单片机的内部复位门槛电压固定，P5. 4不可当复位管脚RST使用，且P3. 2/P3. 3与下载无关，且不支持"程序加密后传输"功能。

建议用户将串口1设在[P3.6/RxD_2,P3.7/TxD_2]或[P1.6/RxD_3, P1.7/TxD_3]

宏晶•STC15W4K60S4系列主要性能：

- 大容量4096字节片内RAM数据存储器
- 高速：1个时钟/机器周期，增强型8051内核，速度比传统8051快7～12倍
  速度也比STC早期的1T系列单片机(如STC12/11/10系列)的速度快20%
- 宽电压：2.4V～5.5V
- 低功耗设计：低速模式，空闲模式，掉电模式(可由外部中断或内部掉电唤醒定时器唤醒)
- 不需外部复位的单片机，ISP编程时16级复位门槛电压可选，内置高可靠复位电路
- 不需外部晶振的单片机，ISP编程时内部时钟从5MHz~35MHz可设(相当于普通8051：60～420MHz)
  内部高精度R/C时钟(±0.3%)，±1%温飘(-40℃~+85℃)，常温下温飘±0.6%(-20℃~+65℃)
- 支持掉电唤醒的资源有：INT0/INT1(上升沿/下降沿中断均可), INT2/INT3/INT4(下降沿中断)：CCP0/CCP1/RxD/RxD2/RxD3/RxD4/T0/T1/T2/T3/T4管脚；内部掉电唤醒专用定时器
- 16/32/40/48/56/60/61K/63.5K字节片内Flash程序存储器，擦写次数10万次以上
- 大容量片内EEPROM功能, 擦写次数10万次以上
- ISP/IAP，在系统可编程/在应用可编程, 无需编程器/仿真器
- 高速ADC，8通道10位，速度可达30万次/秒，8路PWM还可当8路D/A使用
- 比较器，可当1路ADC使用，并可作掉电检测，支持外部管脚CMP+与外部管脚CMP-进行比较，可产生中断，并可在管脚CMPO上产生输出（可设置极性），也支持外部管脚CMP+与内部参考电压进行比较
- 6通道15位专门的高精度PWM(带死区控制) + 2通道CCP (利用它的高速脉冲输出功能可实现2路11~16位PWM)
  ---可用来再实现8路D/A, 或2个16位定时器，或2个外部中断(支持上升沿/下降沿中断)
- 共7个定时器/计数器，5个16位可重装载定时器/计数器(T0/T1/T2/T3/T4,其中T0和T1兼容普通8051的定时器/计数器)，并均可实现时钟输出，另外管脚MCLKO可将内部主时钟对外分频输出（÷1或÷2或÷4），2路CCP可再实现2个定时器
- 可编程时钟输出功能(对内部系统时钟或外部管脚的时钟输入进行时钟分频输出)：
  ① T0在P3.5输出时钟；② T1在P3.4输出时钟；③ T2在P3.0输出时钟；④ T3在P0.4输出时钟； ⑤ T4在P0.6输出时钟，以上5个定时器/计数器输出时钟均可1~65536级分频输出；⑥ 内部主时钟在P5.4/MCLKO对外输出时钟(STC15系列8-pin单片机的主时钟在P3.4/MCLKO对外输出时钟)
- 超高速四串口/UART，四个完全独立的高速异步串行通信端口，分时切换可当9组串口使用
- SPI高速同步串行通信接口
- 硬件看门狗(WDT)
- 先进的指令集结构，兼容普通8051指令集，有硬件乘法/除法指令
- 通用I/O口(62/46/42/38/30/26个)，复位后为：准双向口/弱上拉(8051传统I/O口)
  可设置四种模式：准双向口/弱上拉，强推挽/强上拉，仅为输入/高阻，开漏
  每个I/O口驱动能力均可达到20mA，但整个芯片最大不要超过120mA

选择宏晶•STC15W4K60S4系列单片机理由：

★ 不需外部晶振和外部复位，还可对外输出时钟和低电平复位信号
★ 片内大容量4096字节SRAM
★ 6通道15位专门的高精度PWM(带死区控制) + 2通道CCP (利用它的高速脉冲输出功能可实现2路11~16位PWM)
  ---可用来再实现8路D/A,或2个16位定时器，或2个外部中断(支持上升沿/下降沿中断)
★ 无法解密，宏晶第九代加密技术，现悬赏20万元人民币请专家帮忙查找加密有无漏洞
★ 超强抗干扰：
  1. 高抗静电(ESD保护)整机轻松过2万伏静电测试
  2. 轻松过4kV快速脉冲干扰(EFT测试)
  3. 宽电压，不怕电源抖动
  4. 宽温度范围，-40℃~+85℃
★ 大幅降低EMI，内部可配置时钟，1个时钟/机器周期，可用低频时钟
  ---出口欧美的有力保证
★ 超低功耗：
  1. 掉电模式：外部中断唤醒功耗 <0.1uA
  2. 空闲模式：典型功耗 <1mA，
  3. 正常工作模式：4mA ~ 6mA
  4. 掉电模式可由外部中断或内部掉电唤醒专用定时器唤醒，适用于电池供电系统，如水表、气表等
★ 在系统可仿真，在系统可编程，无需专用编程器，无需专用仿真器，可远程升级
★ 可送USB型联机/脱机下载烧录工具STC-U7(人民币100元)，1万片/人/天，有自动烧录机接口

宏晶科技，中国大陆本土第一家战胜全球所有竞争对手的MCU设计公司，北京加油！